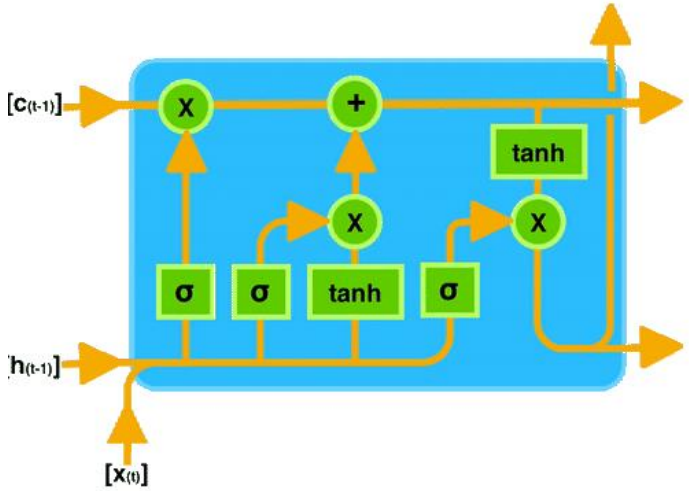
上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

RNNs and LSTMs

(2026)

Lecturer: Xue Yang

yangxue.site



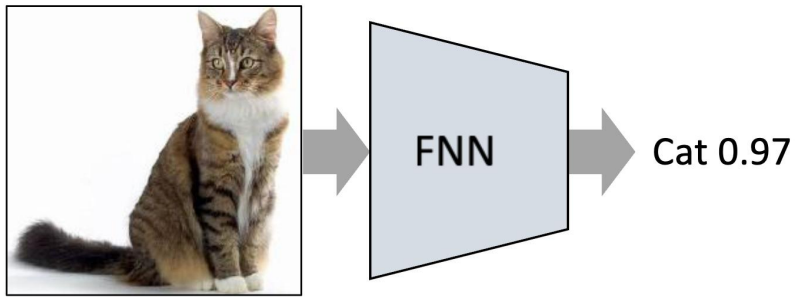
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

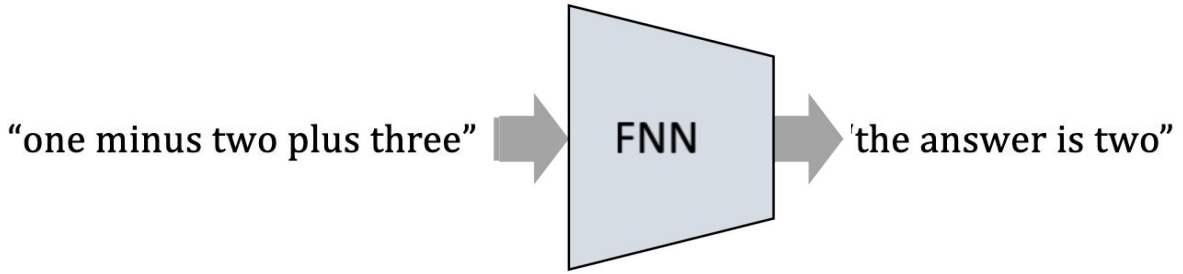
Contents

- **Word Representation**
- Recurrent Neural Networks
- Long Short-Term Memory Networks
- Sequence To Sequence

Motivation

- There are many **time-series data sets**, such as language, video, and bio-signals that could not fit into this framework.

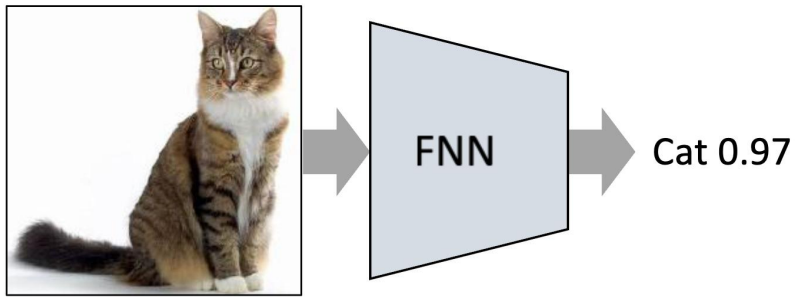




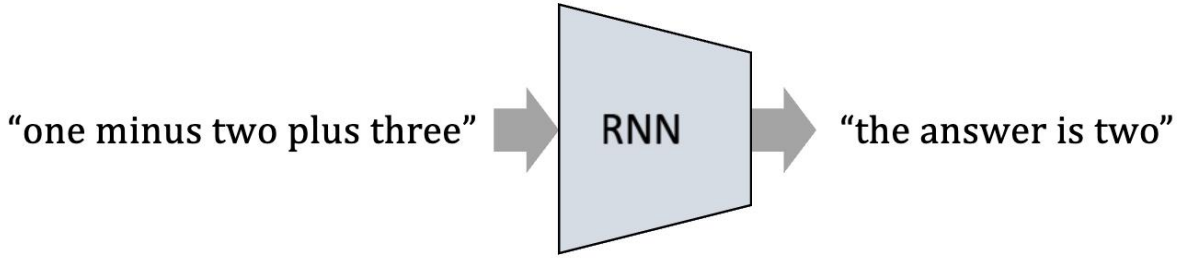
Great! But …

Motivation

- There are many **time-series data sets**, such as language, video, and bio-signals that could not fit into this framework.
- The **Recurrent Neural Network (RNN)** is a deep learning architecture designed for processing time-series data.



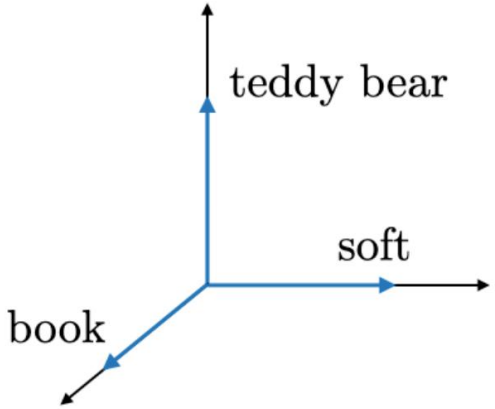
Great! But …



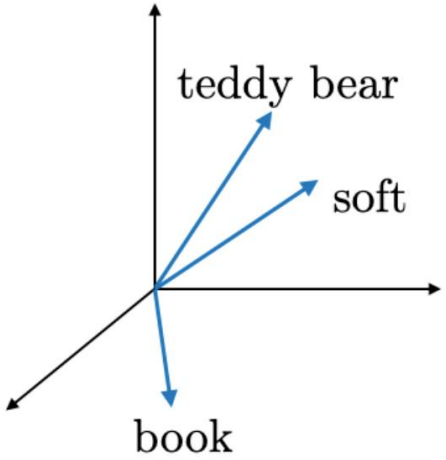
today's main character

How to represent word?

- **One Solution**: Words as vector



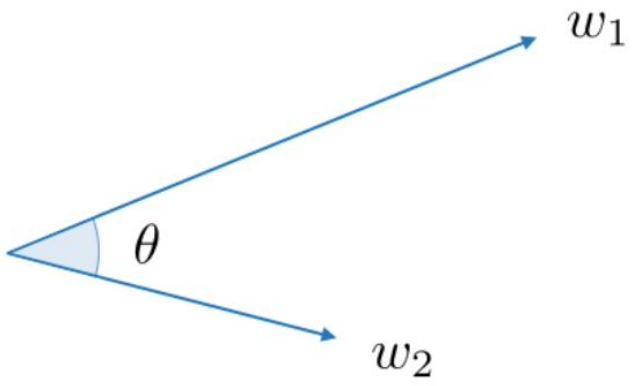
Sparse vector



Dense vector

How to represent word?

- **One Solution**: Words as vector
- We can compute the difference between each words by cosine similarity



$$\text{similarity} = \frac{w_1 \cdot w_2}{||w_1||\ ||w_2||} = \cos(\theta)$$

How to transfer the words into vectors?

- hotel, conference, motel – a localist representation

motel = [? ? ?]
hotel = [? ? ?]
conference = [? ? ?]

Sparse solution: one-hot vectors

- **hotel, conference, motel – a localist representation**

Means one 1, the rest 0s

Such symbols for words can be represented by **one-hot** vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

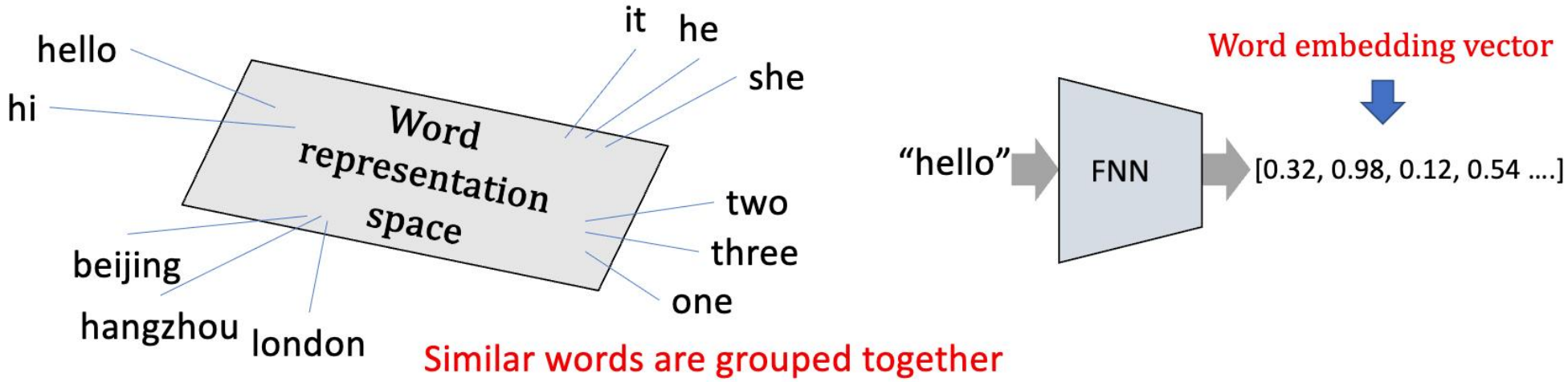
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

conference = [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

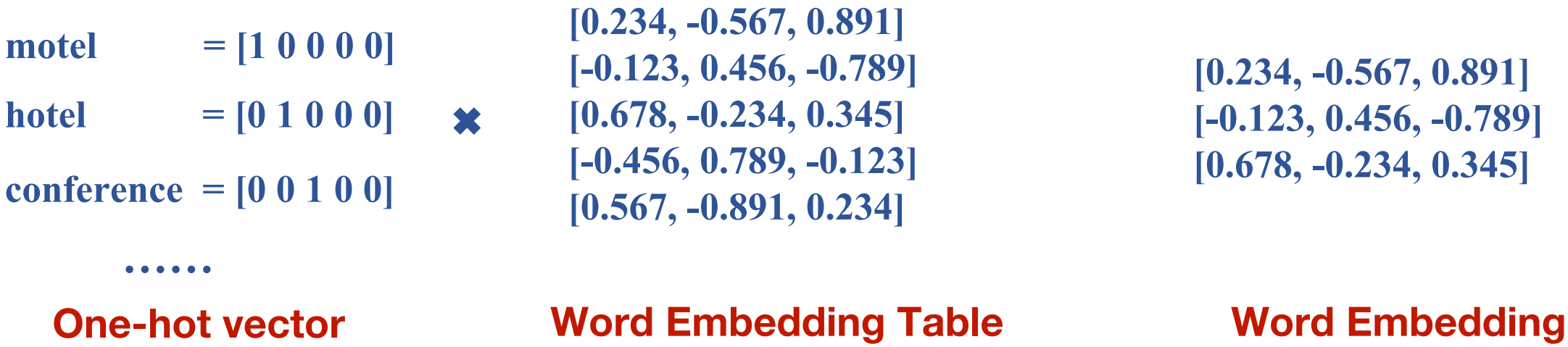
Dense solution: Word Embedding

- We can also represent a word using a vector of floating-point numbers.



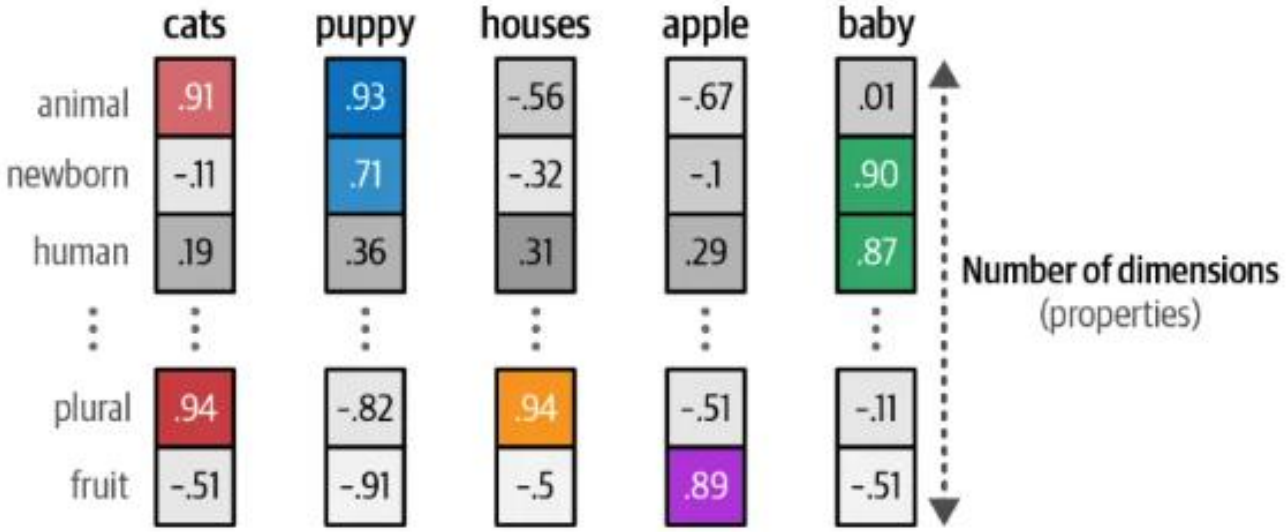
Dense solution: Word Embedding

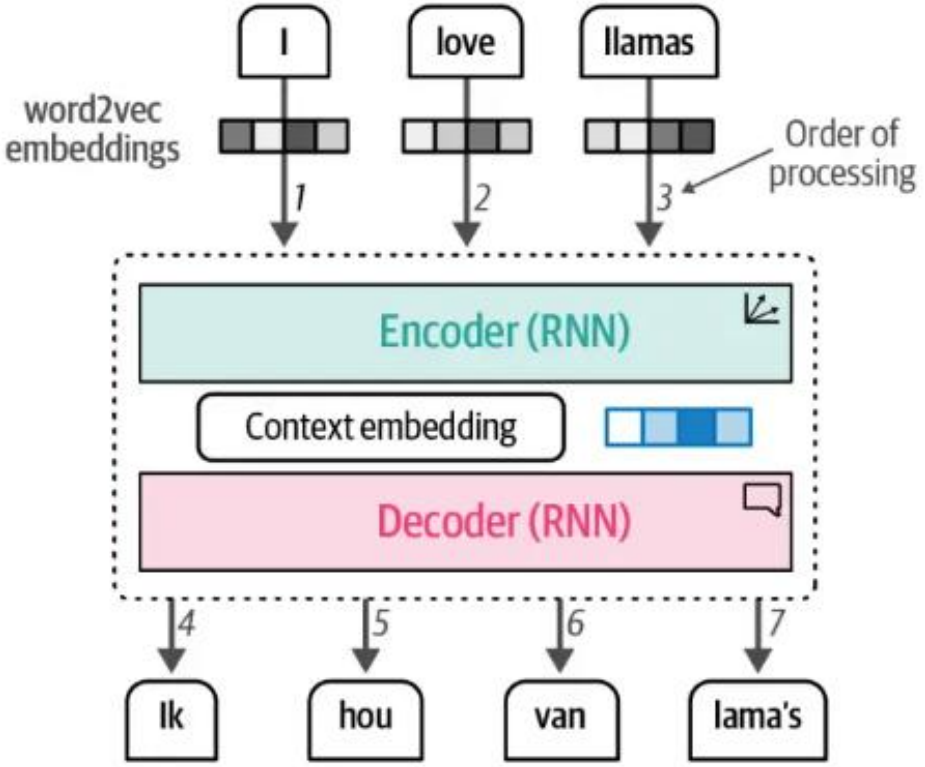
- Given a vocabulary with 5 words, we can have a word embedding table that each word has 3 (Dimension) feature values.



Vector dimension = number of feature (Better than sparse solution)

Dense solution: Word Embedding





How to get word vectors?

expect =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

How to get word vectors?

- **Distributional semantics**: **A word's meaning is given by the words that frequently appear close-by**.
- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of w to build up a representation of w

…government debt problems turning into **banking** crises as happened in 2009…
…saying that Europe needs unified **banking** regulation to replace the hodgepodge…
…India has just given its **banking** system a shot in the arm…

These **context words** will represent **banking**

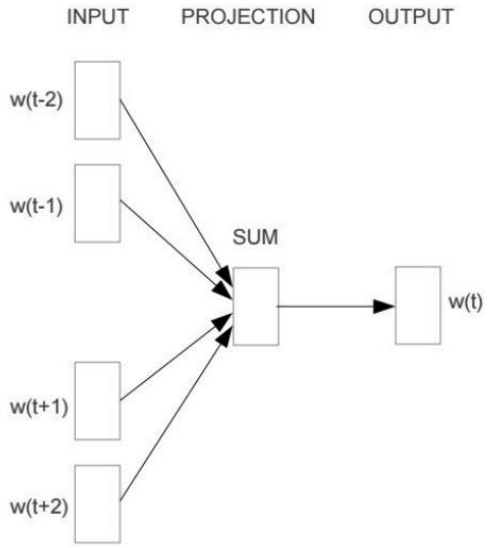
How to get word vectors?

- **Word2vec** (Mikolov et al. 2013) is a framework for learning word vectors

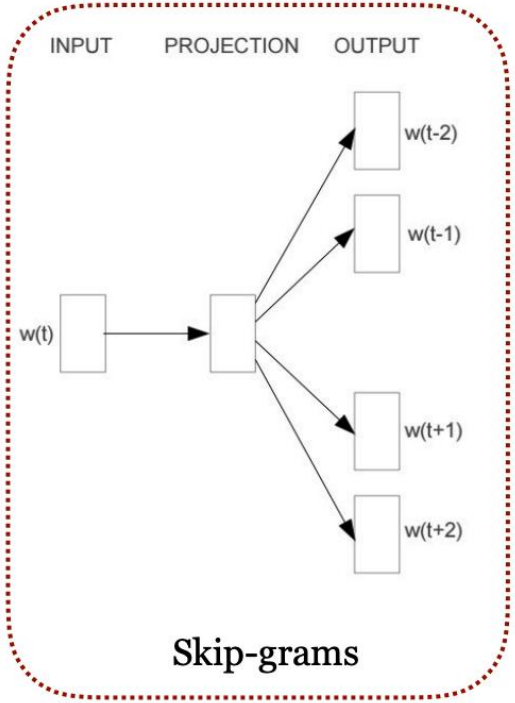
- Two model variants:
  - ✓ **Skip-grams (SG)**

    Predict context ("outside") words (position independent) given center word
  - ✓ **Continuous Bag of Words (CBOW)**

    Predict center word from (bag of) context words



Continuous Bag of Words (CBOW)



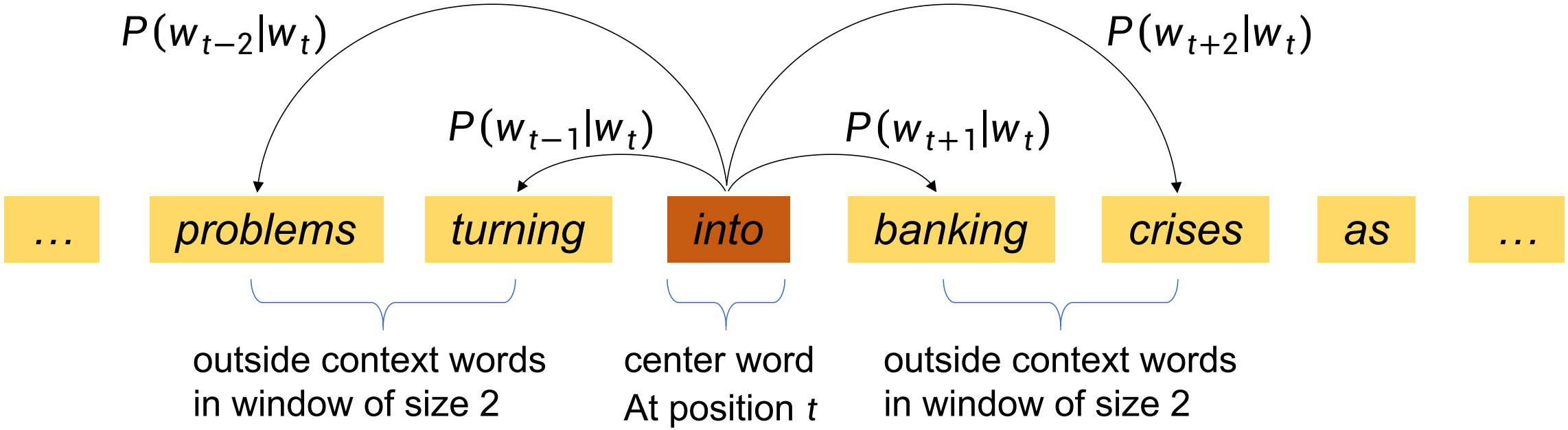
Skip-grams

How to get word vectors?

- **Word2vec** (Mikolov et al. 2013) is a framework for learning word vectors
- **Idea**:
  - ✓ We have a large corpus ("body") of text: a long list of words.
  - ✓ Every word in a fixed vocabulary is represented by a **vector**.
  - ✓ Go through each position t in the text, which has a center word c and context ("outside") words o.
  - ✓ Use the **similarity of the word vectors** for c and o to calculate the probability of o given c (or vice versa)
  - ✓ **Keep adjusting the word vectors** to maximize this probability

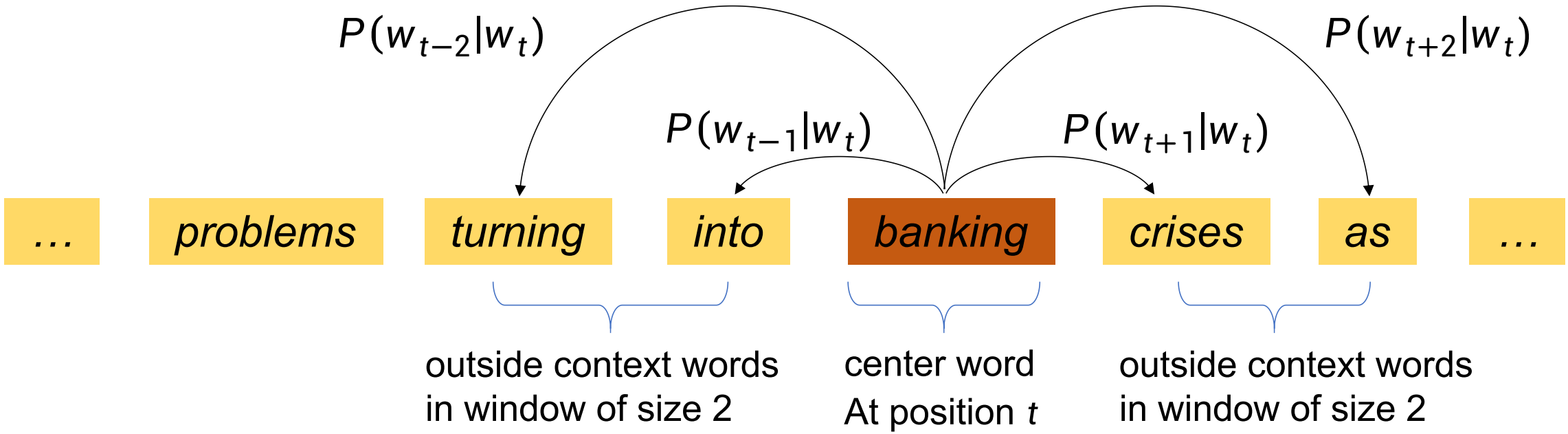
Word2vec

- Example windows and process for computing: $P(w_{t+j}|w_t)$



Word2vec

- Example windows and process for computing: $P(w_{t+j}|w_t)$



Word2vec: objective function

- For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\textbf{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

Word2vec: objective function

- For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\textbf{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

- The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P(w_{t+j}|w_t;\theta)$$

- **Question**: How to calculate $P(w_{t+j}|w_t;\theta)$?

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P(w_{t+j}|w_t;\theta)$$

- **Question**: How to calculate $P(w_{t+j}|w_t;\theta)$?
- **Answer**: We will use *two* vectors per word w :
  - ✓ v_w when w is a center word.
  - ✓ u_w when w is a context word.

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P(w_{t+j}|w_t;\theta)$$

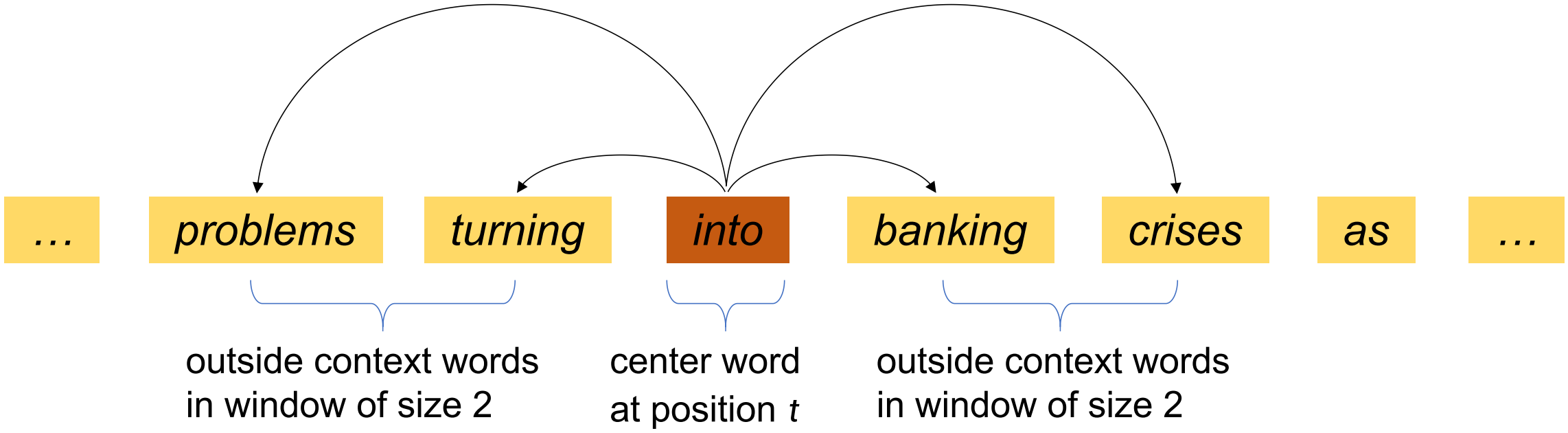
- **Question**: How to calculate $P(w_{t+j}|w_t;\theta)$?
- **Answer**: We will use *two* vectors per word w :
  - ✓ v_w when w is a center word.
  - ✓ u_w when w is a context word.
- Then for a center word c and a context word o:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

also called prediction function

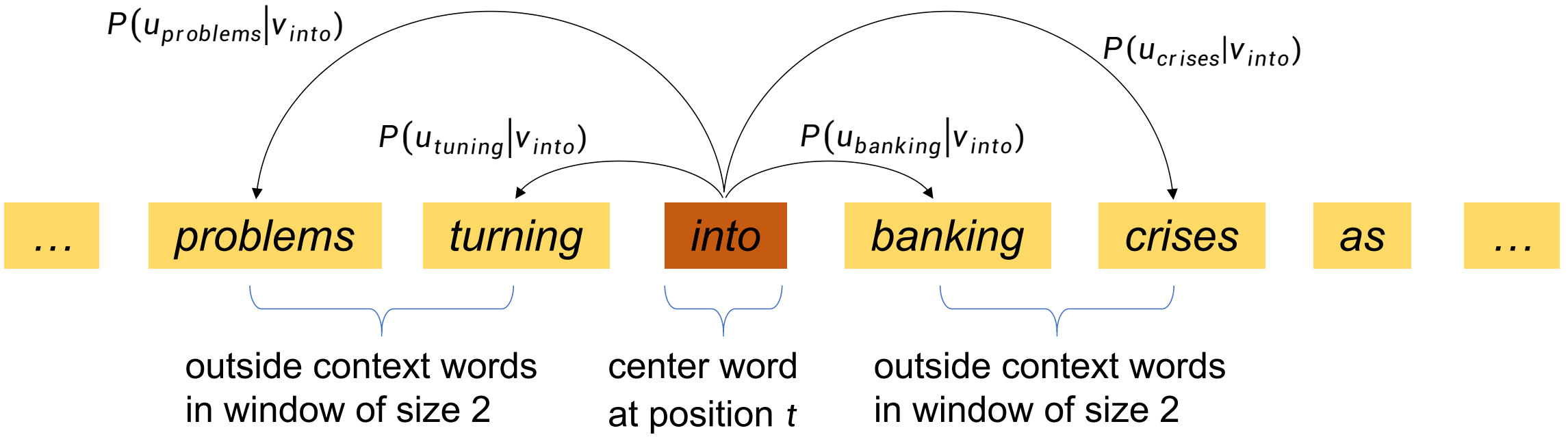
Word2vec with Vectors

- Example windows and process for computing $P(w_{t+j}|w_t)$
- $P(u_{problems}|v_{into})$ short for $P(problems|into; u_{problems}, v_{into}, \theta)$



Word2vec with Vectors

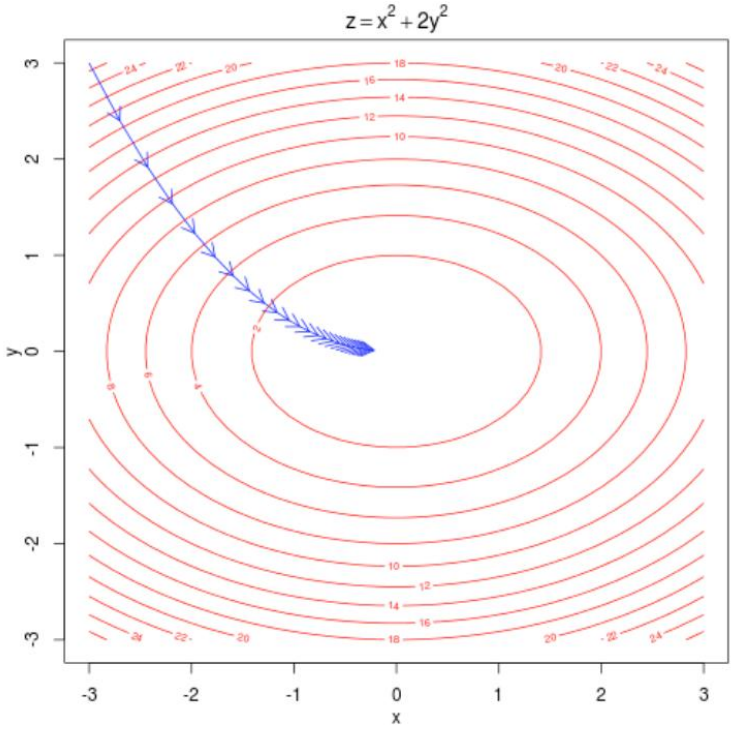
- Example windows and process for computing $P(w_{t+j}|w_t)$
- $P(u_{problems}|v_{into})$ short for $P(problems|into; u_{problems}, v_{into}, \theta)$



Word2vec: optimization

- To train a model, we gradually adjust parameters to minimize a loss
- In our case, with d-dimensional vectors and V-many words, we have:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



Word2vec

- The normalization term is computationally expensive (when many output classes)

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

A big sum over words

Word2vec

- The normalization term is computationally expensive (when many output classes)

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

A big sum over words

- Hence, in standard word2vec and HW2 you implement the skip-gram model with **negative sampling**.

Word2vec

- The normalization term is computationally expensive (when many output classes)

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

A big sum over words

- Hence, in standard word2vec and HW2 you implement the skip-gram model with **negative sampling**.
- **Main idea**: train **binary logistic regressions** to differentiate a true pair (center word and a word in its context window) versus several "noise" pairs (the center word paired with a random word)

Word2vec

- Overall objective function (they maximize): $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

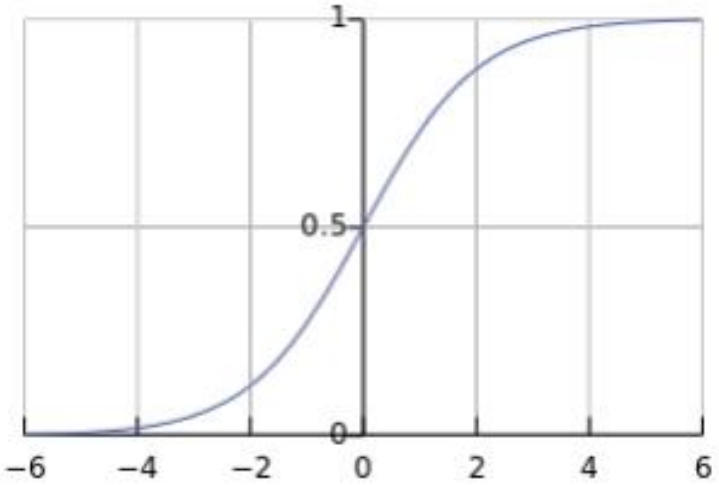
$$J(\theta) = \log\sigma(u_o^T v_c) + \sum_{j=1}^{k} \mathbb{E}_{j\sim P(\omega)}[\log\sigma(-u_j^T v_c)]$$

Word2vec

- Overall objective function (they maximize): $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

$$J(\theta) = \log\sigma(u_o^T v_c) + \sum_{j=1}^{k} \mathbb{E}_{j\sim P(\omega)}[\log\sigma(-u_j^T v_c)]$$

- The logistic/sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$



- We maximize the probability of two words co-occurring in first log and minimize probability of noise words in second part.

Word2vec

- Using notation consistent with this class and HW2:

$$J_{neg-sample}(u_o, v_c, U) = -\log\sigma(u_o^T v_c) - \sum_{k\in\{K\ sampled\ indices\}} \log\sigma(u_k^T v_c)$$

Word2vec

- Using notation consistent with this class and HW2:

$$J_{neg-sample}(u_o, v_c, U) = -\log\sigma(u_o^T v_c) - \sum_{k\in\{K\ sampled\ indices\}} \log\sigma(u_k^T v_c)$$

- We take k negative samples (using word probabilities)
- Maximize probability that real outside word appears; minimize probability that random words appear around center word

Word2vec

- Using notation consistent with this class and HW2:

$$J_{neg-sample}(u_o, v_c, U) = -\log\sigma(u_o^T v_c) - \sum_{k\in\{K\ sampled\ indices\}} \log\sigma(u_k^T v_c)$$

- We take k negative samples (using word probabilities)
- Sample with $P(w) = \frac{U(w)^{3/4}}{Z}$, the unigram distribution $U(w)$ raised to the 3/4 power. The power makes less frequent words be sampled more often

  e.g. 0.9^(0.75) ≈ 0.93; 0.01^(0.75) ≈ 0.056

Word2vec

- We iteratively take gradients at each window for SGD

- In each window, we only have at most $2m + 1$ words plus $2km$ negative words with negative sampling, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

Word senses and word sense ambiguity

- Most words have lots of meanings!
  - ✓ Especially common words
  - ✓ Especially words that have existed for a long time

- Example: **pike**

- *Does one vector capture all these meanings or do we have a mess?*

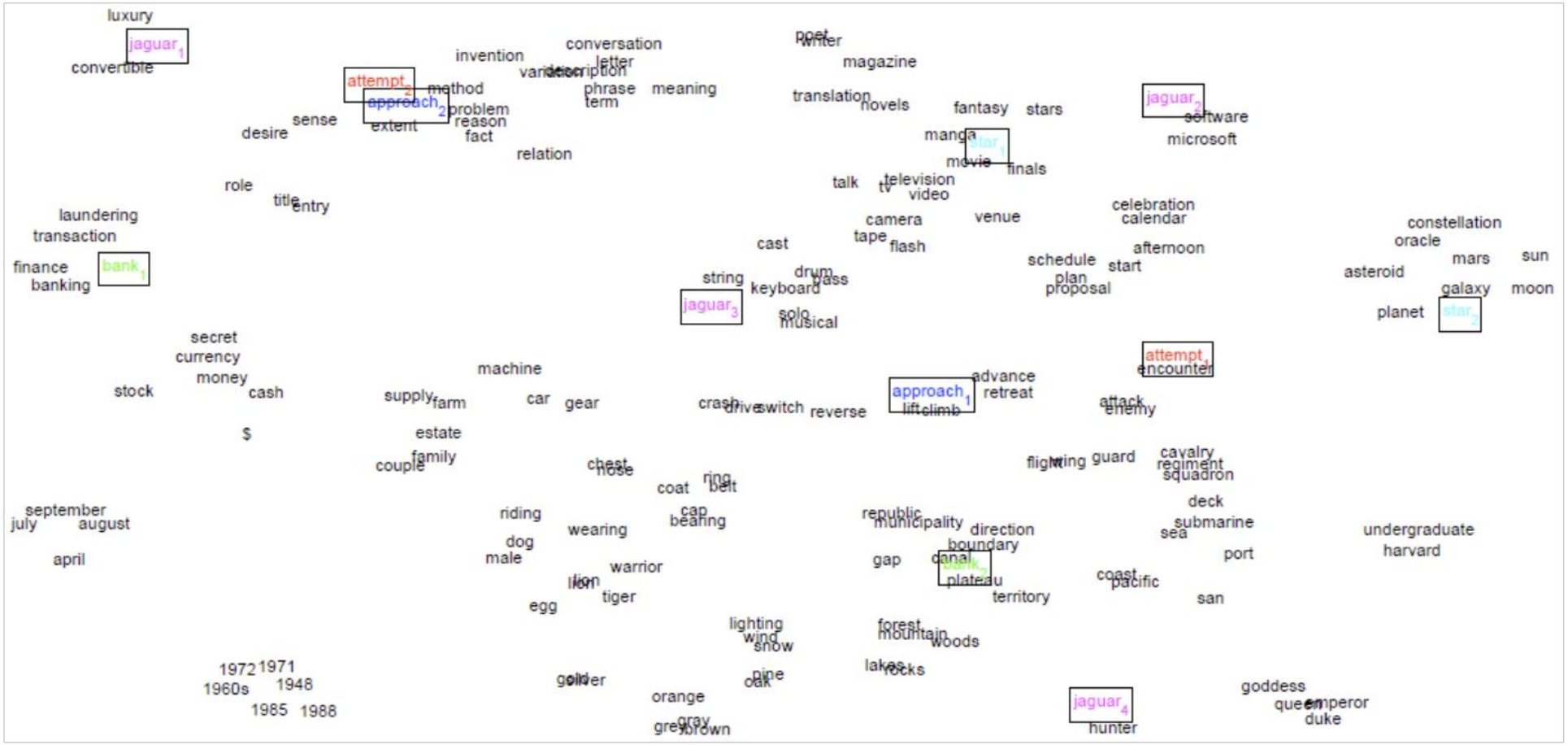
Word senses and word sense ambiguity

- Example: **pike**
  - ✓ A sharp point or staff
  - ✓ A type of elongated fish
  - ✓ A railroad line or system
  - ✓ A type of road
  - ✓ The future (coming down the pike)
  - ✓ A type of body position (as in diving)
  - ✓ To kill or pierce with a pike
  - ✓ To make one's way (pike along)
  - ✓ In Australian English, pike means to pull out from doing something: I reckon he could have climbed that cliff, but he piked!

- 尖头或棒
- 细长鱼的一种
- 铁路线或系统
- 道路的一种
- 未来（顺势而下）
- 一种身体姿势（如潜水）
- 用长矛杀死或刺穿
- 开路（一路前行）
- 在澳大利亚英语中，pike 的意思是从做某事中退出：我认为他本可以爬上那个悬崖，但他 pike!

Word Representation

- **Idea**: Cluster word windows around words, retrain with each word assigned to multiple different clusters bank1, bank2, etc.



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Contents

- Word Representation
- **Recurrent Neural Networks**
- Long Short-Term Memory Networks
- Sequence To Sequence

Task Description

- Given a sequence of words $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(t)}$ compute the probability distribution of the next word

$$P(x^{(t+1)}|x^{(t)}, \ldots, x^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \ldots, w_{|V|}\}$

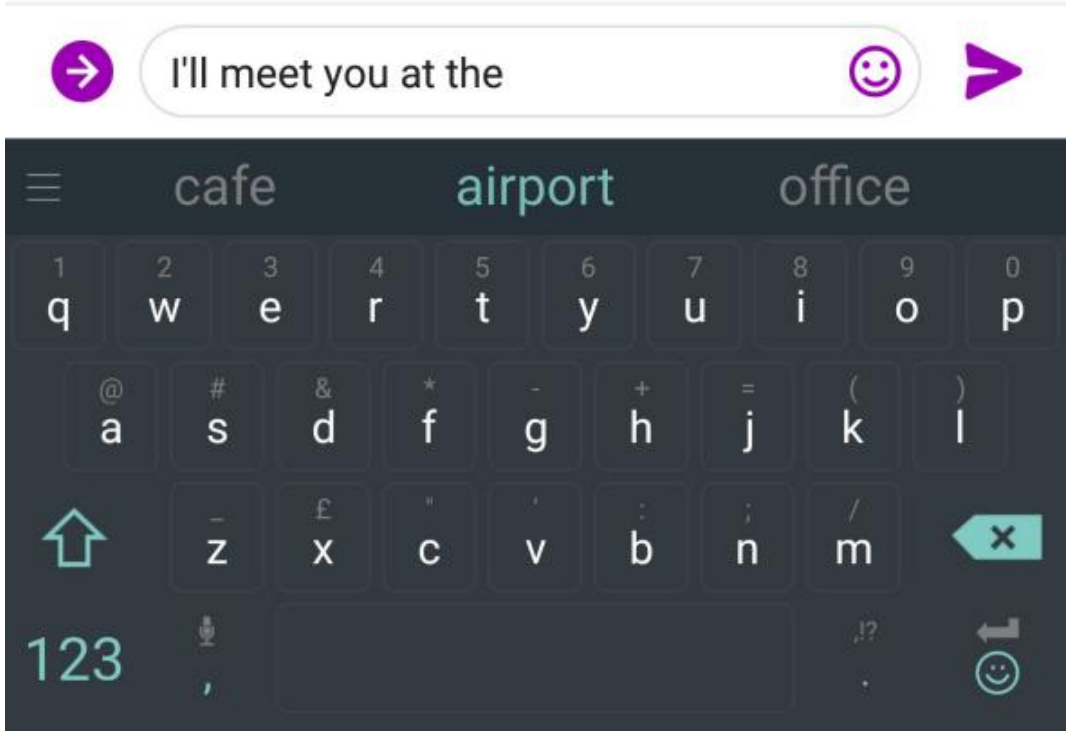
- You can also think of the language model of next word prediction as a system that assigns a probability to a piece of text.

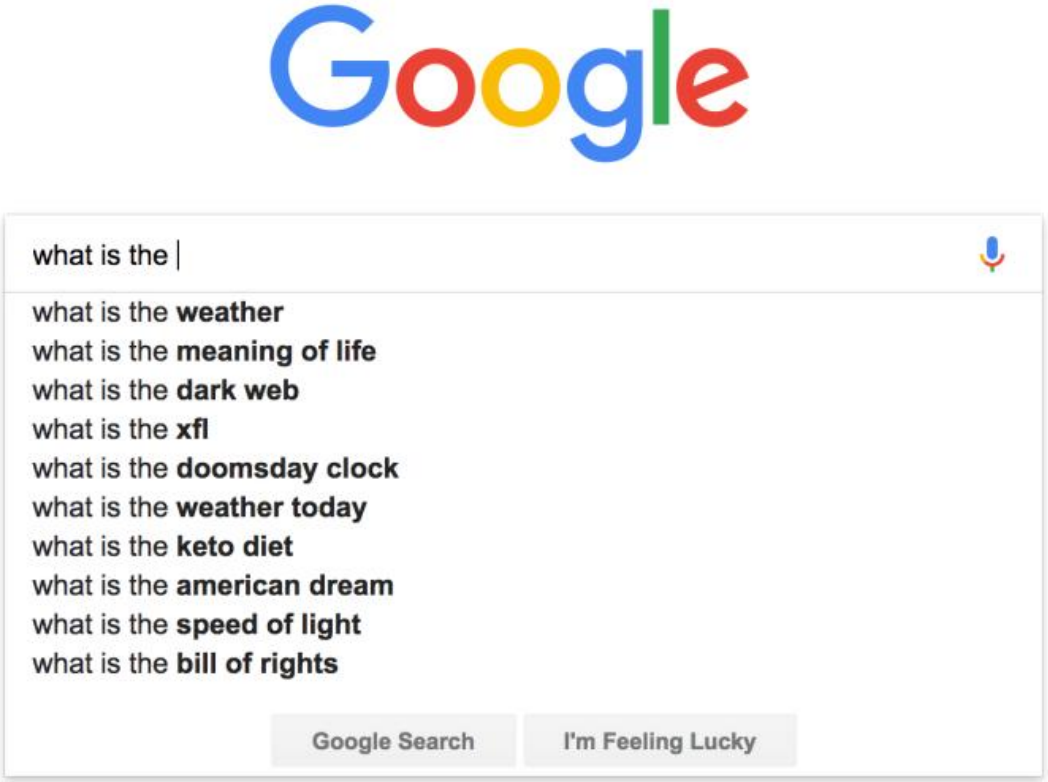
$$P(x^{(1)}, \ldots, x^{(t)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times \ldots \times P(x^{(T)}|x^{(T-1)}, \ldots, x^{(1)})$$
$$= \prod_{t=1}^{T} P(x^{(t)}|x^{(t-1)}, \ldots, x^{(1)})$$

This is what our Language Model (LM) provides

Next Word Prediction Model

- Input an unfinished sentence. A language model tries to predict next word.



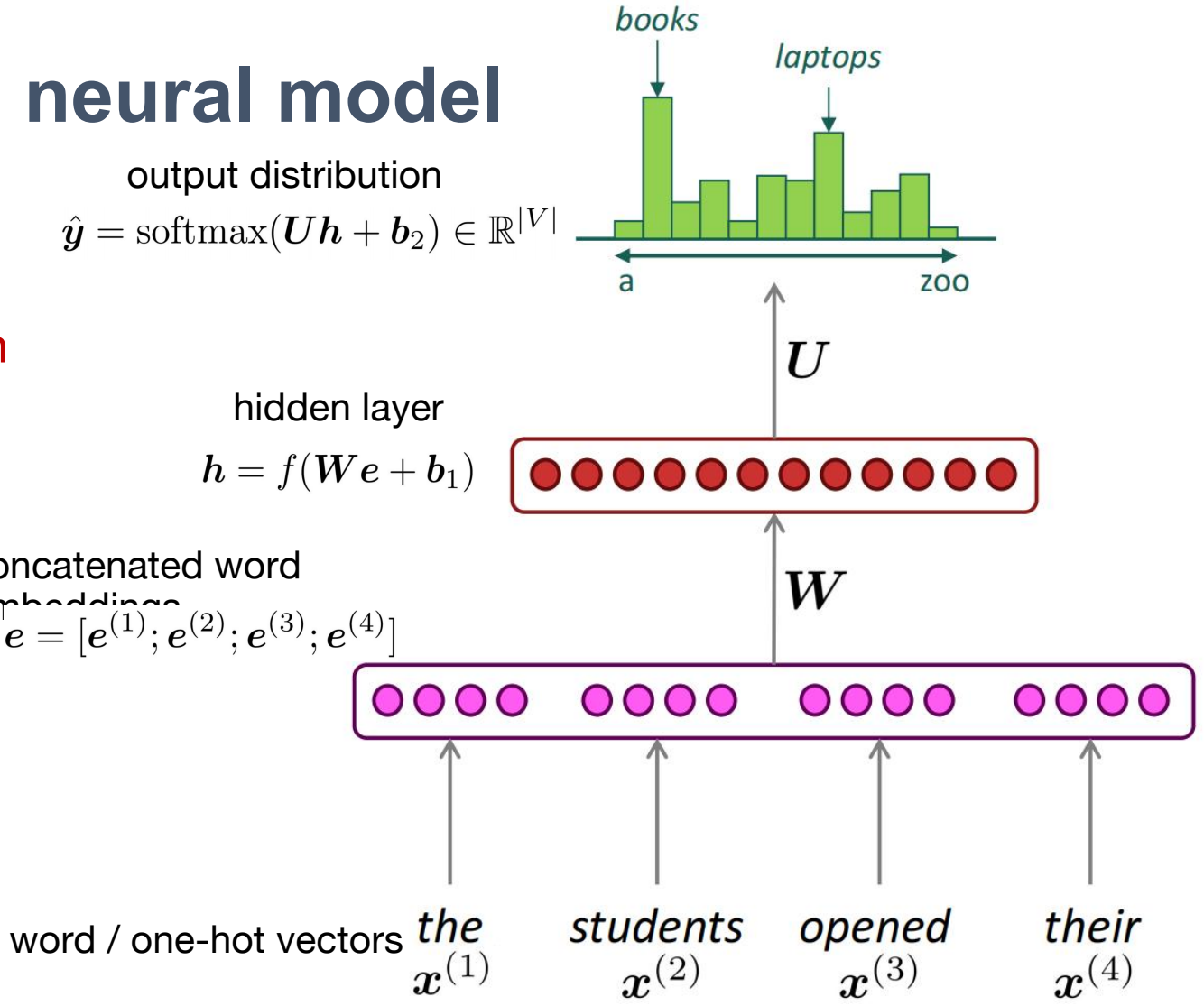


A (fixed) window-based neural model

- Disadvantages
- ✓ Influenced by fix window size
  small window → weak representation
  enlarge window → Window can never be large enough!

- ✓ $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W.
  No symmetry in how the inputs are processed.

We need a class of neural networks allowing to handle variable length inputs

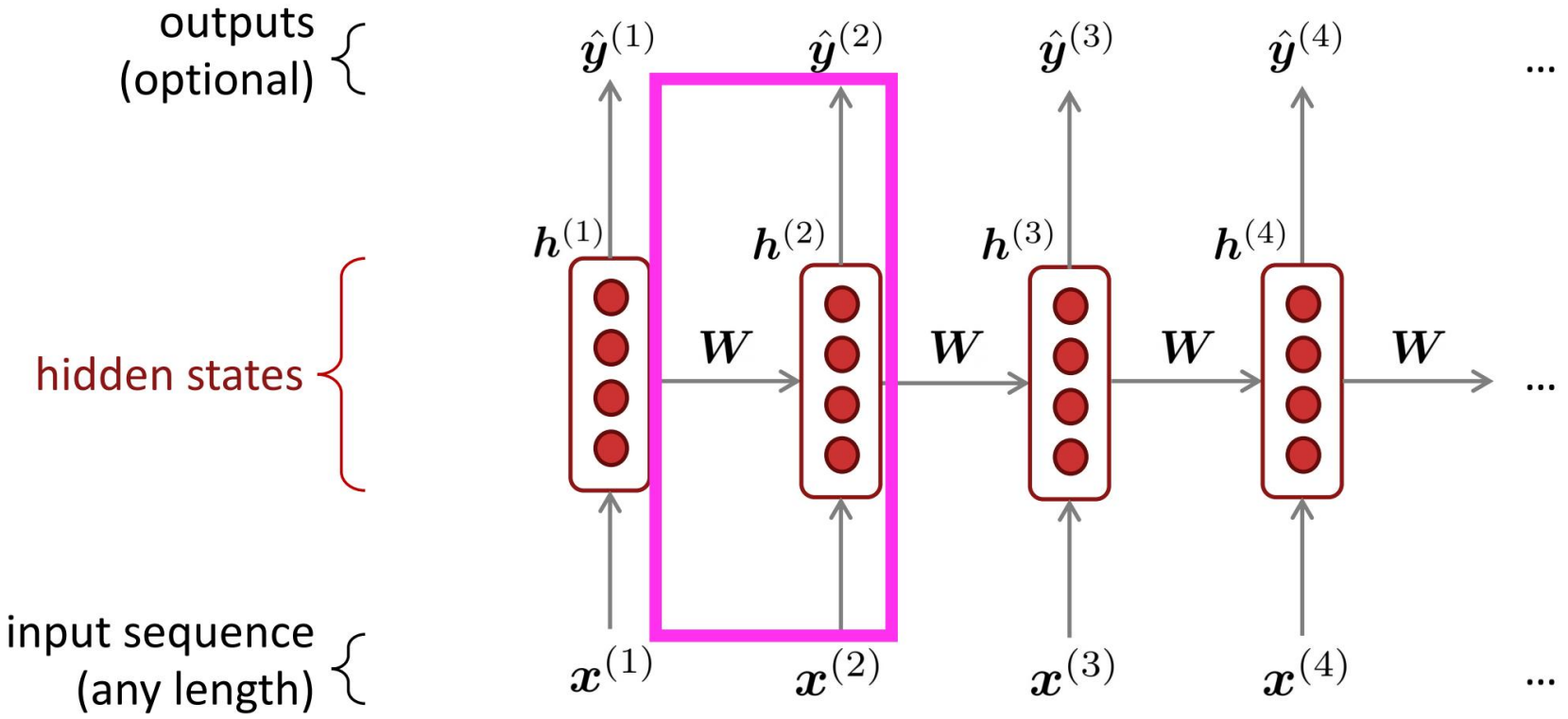


Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

Recurrent Neural Networks

We need a class of neural networks allowing to handle variable length inputs

Core idea: Apply the same weights *W* repeatedly



Recurrent Neural Networks

✓ **output distribution**

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

✓ **hidden layer**

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

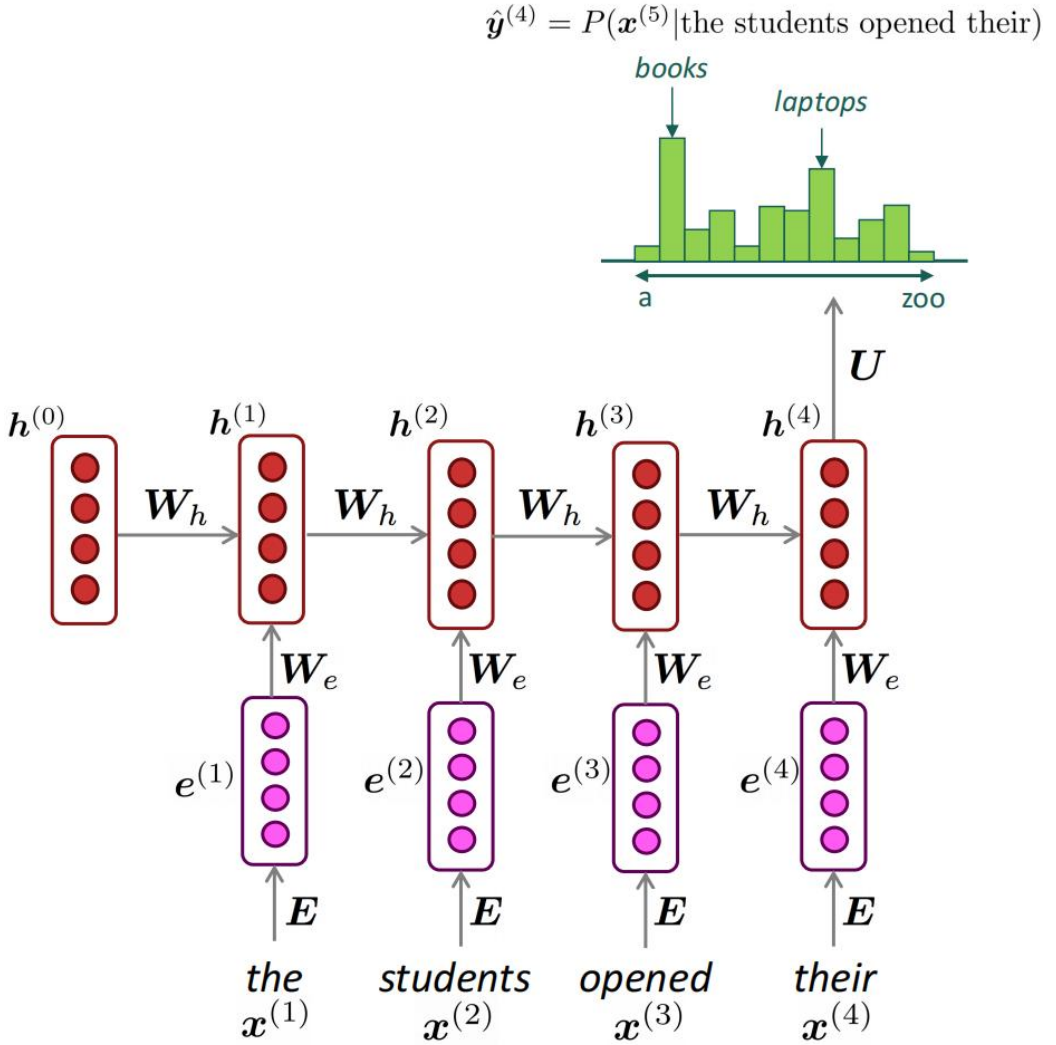
$\boldsymbol{h}^{(0)}$ is the initial hidden state

✓ **word embeddings**

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

✓ **word / one-hot**

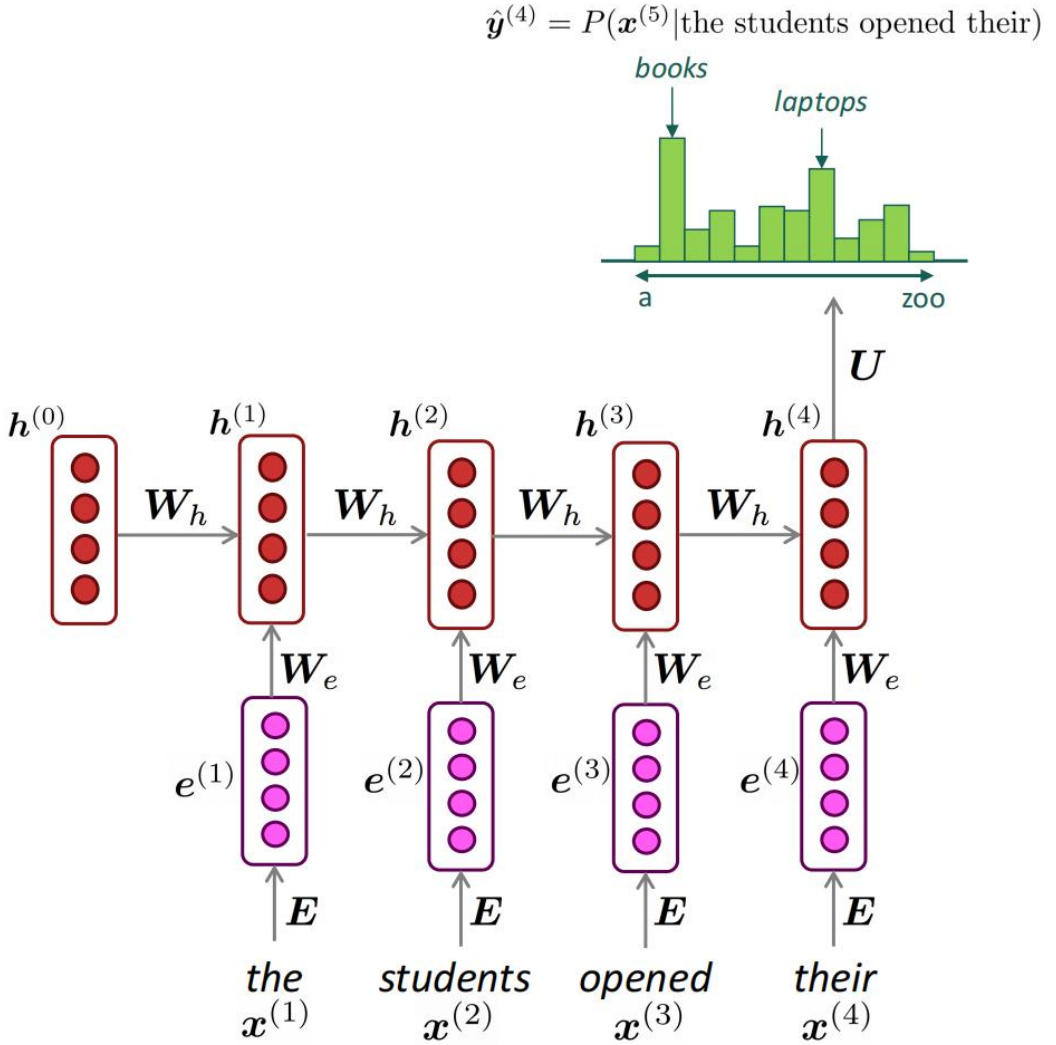
$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Recurrent Neural Networks

- RNN Advantages
  - ✓ Can process **any length** input.
  - ✓ Computation for step *t* can (in theory) use information from **many steps back**
  - ✓ **Model size doesn't increase** for longer input context
  - ✓ Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.

- RNN Disadvantages
  - ✓ Recurrent computation is **slow**
  - ✓ In practice, difficult to access information from **many steps back**.



Training an RNN Language Models

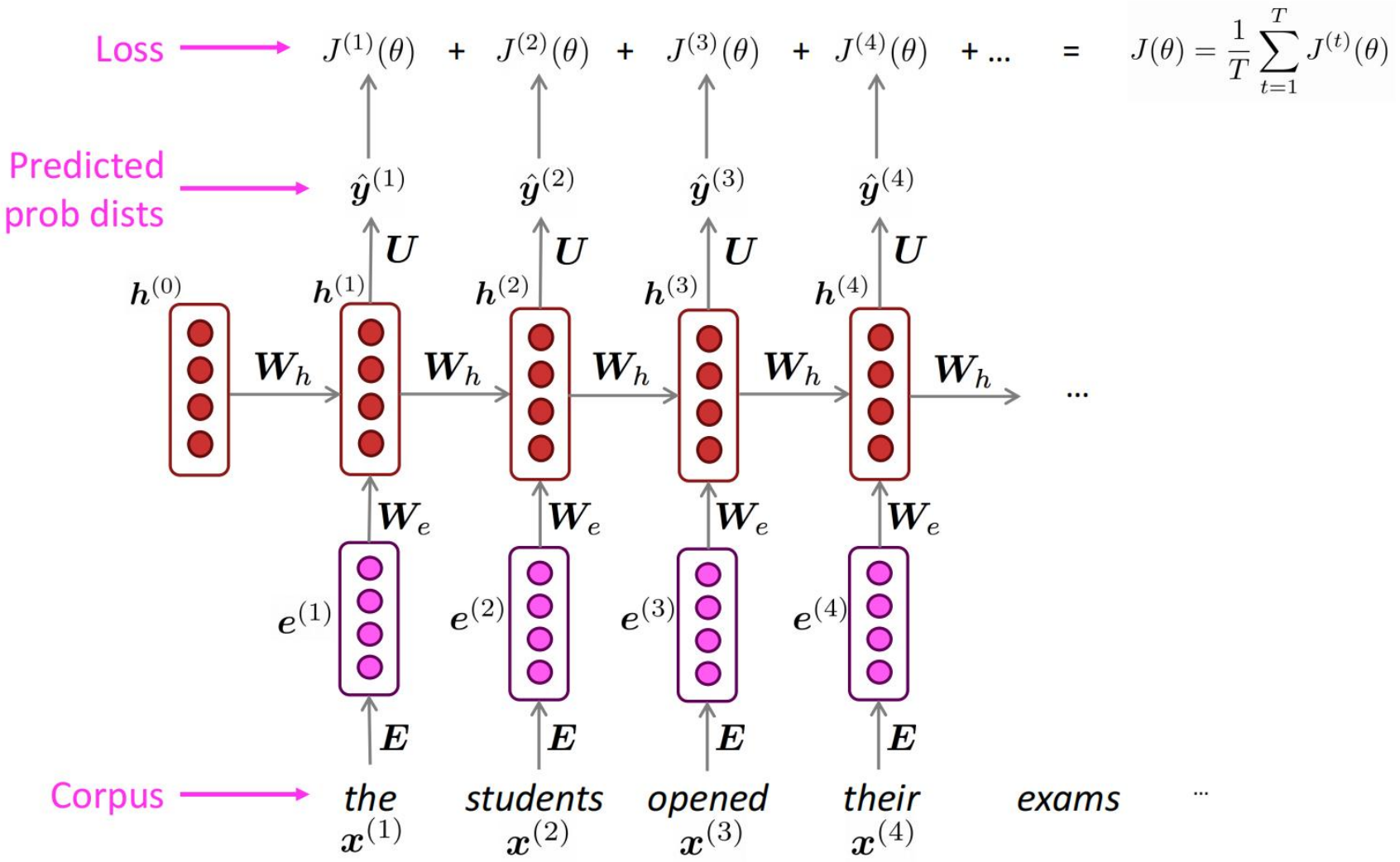
- Get a big corpus of text which is a sequence of words $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(t)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
  - i.e., predict probability dist of every word, given words so far.
- Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$ and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

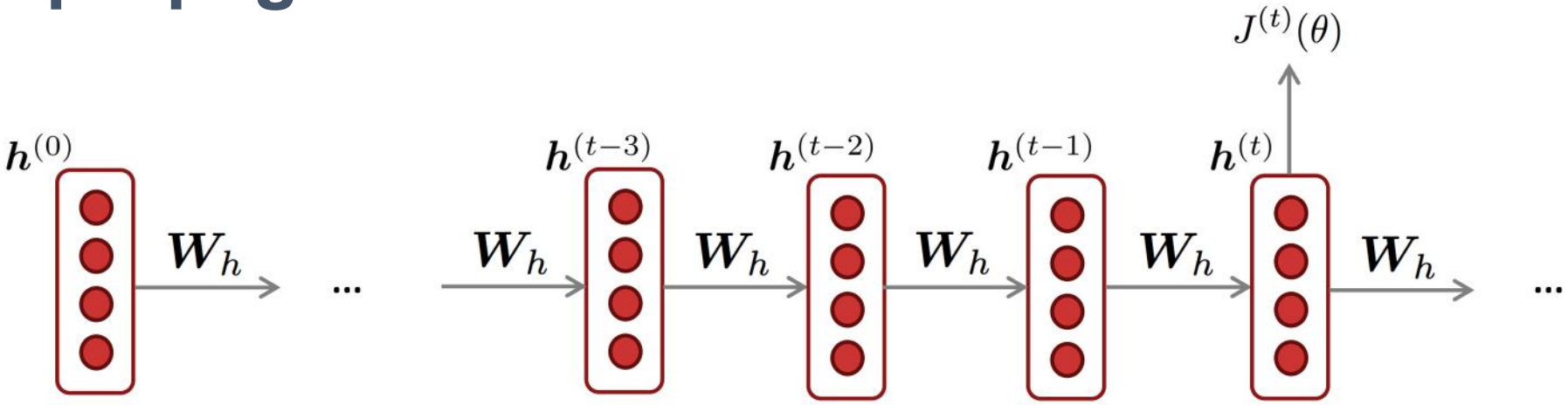
- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

Recurrent Neural Networks



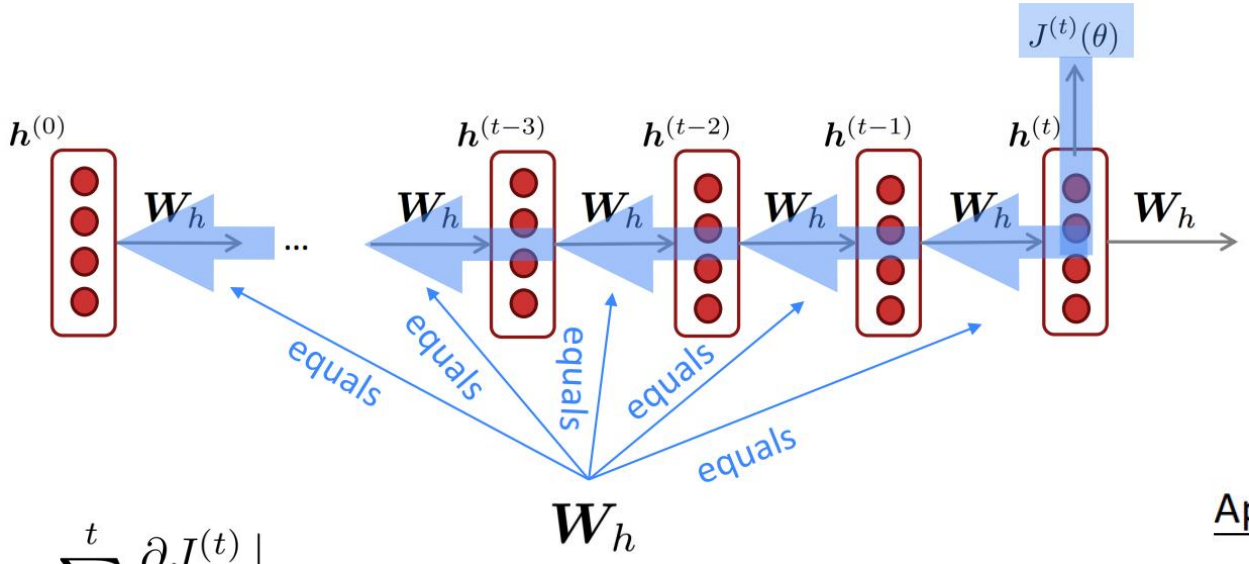
Backpropagation for RNNs



The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears

- **Question**: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix W_h?
- **Answer**: $\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$

Backpropagation for RNNs: Backpropagation Through Time



$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}$$

- Backpropagate over timesteps i = t, … ,0, summing gradients as you go. This algorithm is called “backpropagation through time”

- In practice, often “truncated” after ~20 timesteps for training efficiency reasons

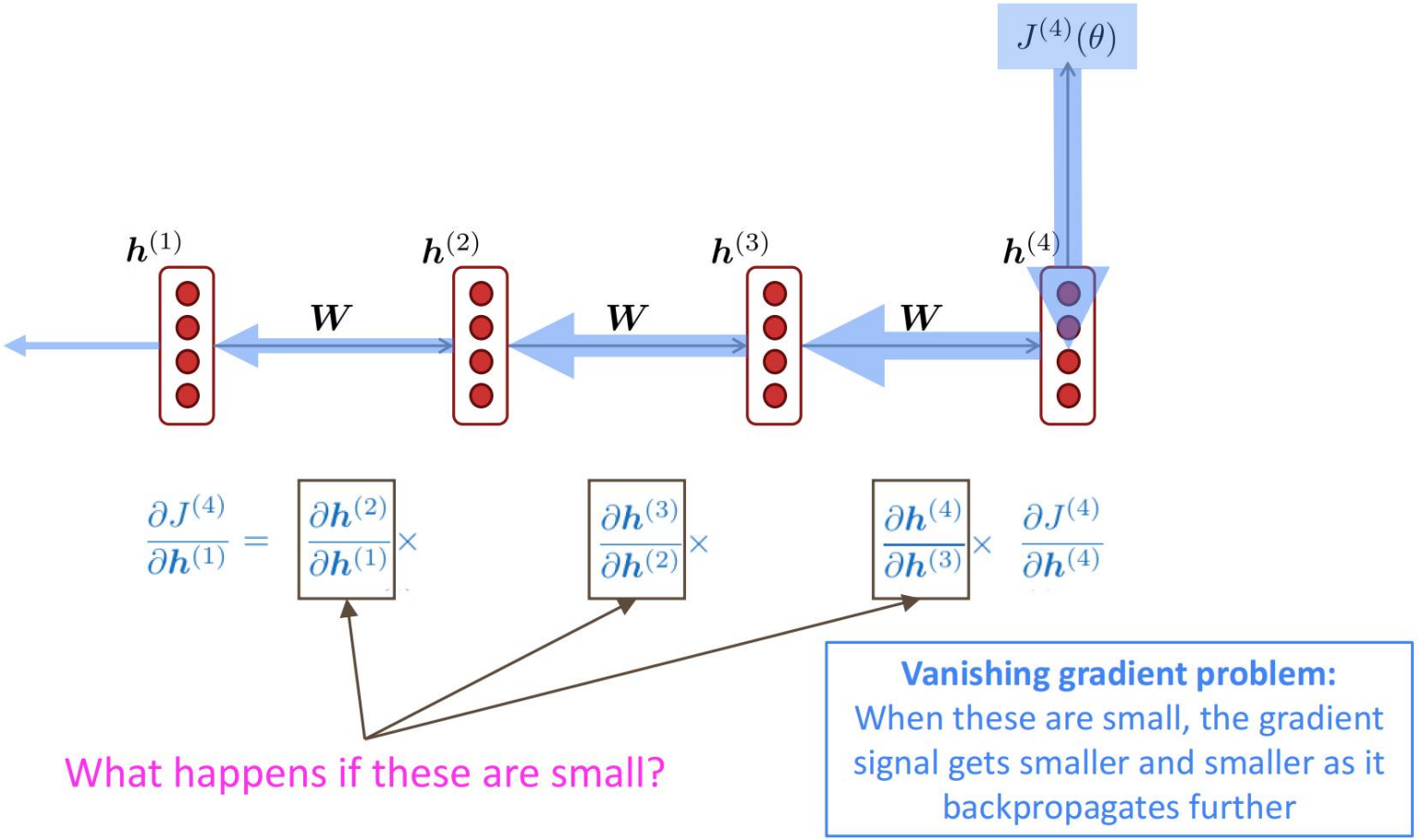
Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} \frac{\partial \boldsymbol{W}_h|_{(i)}}{\partial \boldsymbol{W}_h}$$

($\frac{\partial \boldsymbol{W}_h|_{(i)}}{\partial \boldsymbol{W}_h} = 1$)

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)}$$

Problems with RNNs: vanishing gradient



Problems with RNNs: exploding gradient

- The gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a weird and bad parameter configuration (with large loss)

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

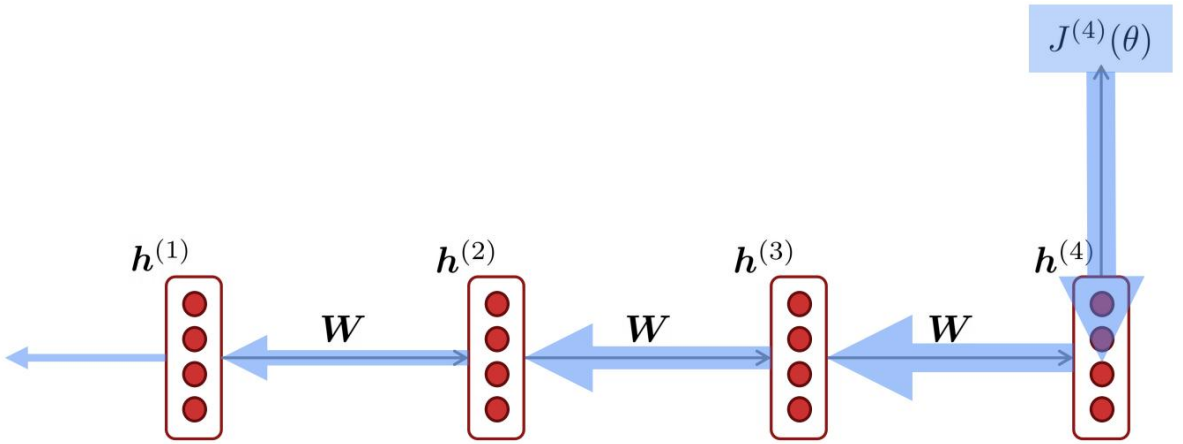
Solutions

- The gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update
- **Intuition**: take a step in the same direction, but a smaller step

How to fix the vanishing gradient problem?



- The main problem is that it's too difficult for the RNN to learn to **preserve information over many timesteps.**
- In a vanilla RNN, the **hidden state is constantly being rewritten**

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

- Creating more direct and linear pass-through connections in model
  Attention, residual connections, etc.

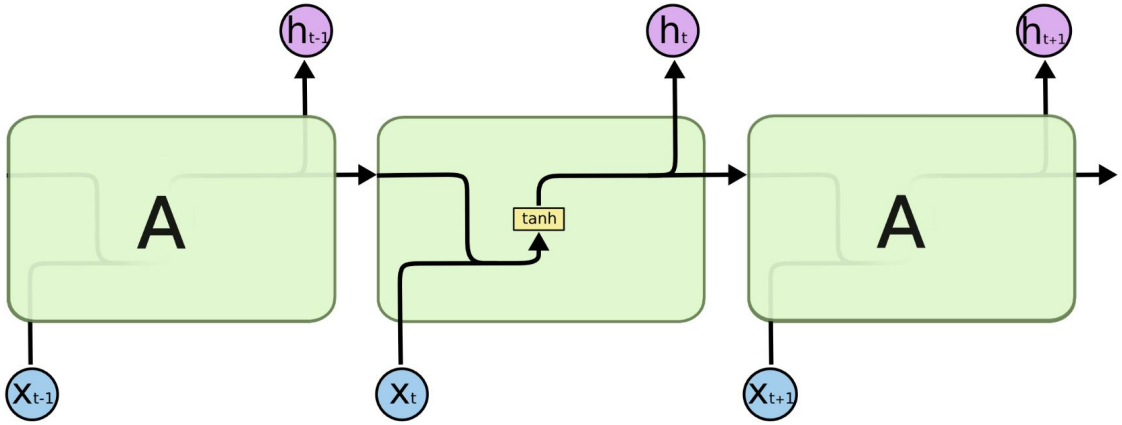
Could we design an RNN with separate memory which is added to?

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

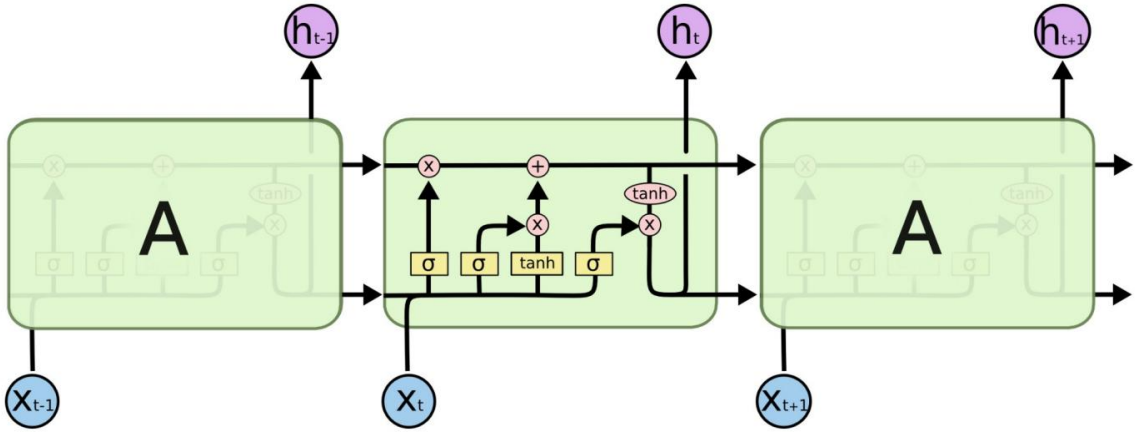
Contents

- Word Representation
- Recurrent Neural Networks
- **Long Short-Term Memory Networks**
- Sequence To Sequence

RNN vs LSTM



Vanilla RNNs

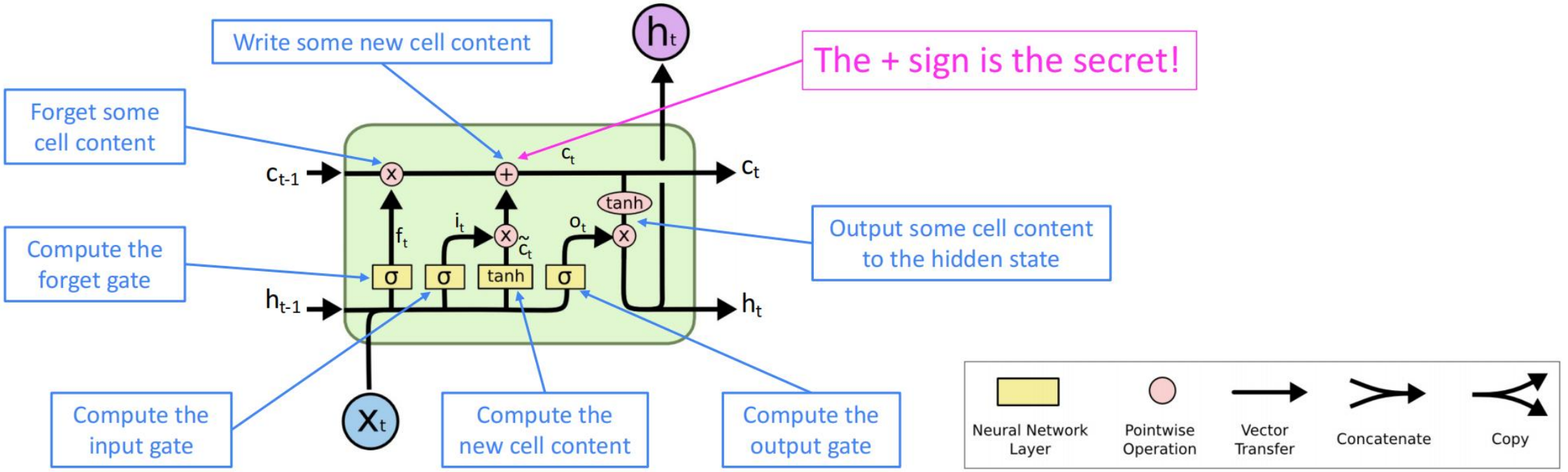


LSTM

Long Short-Term Memory Networks

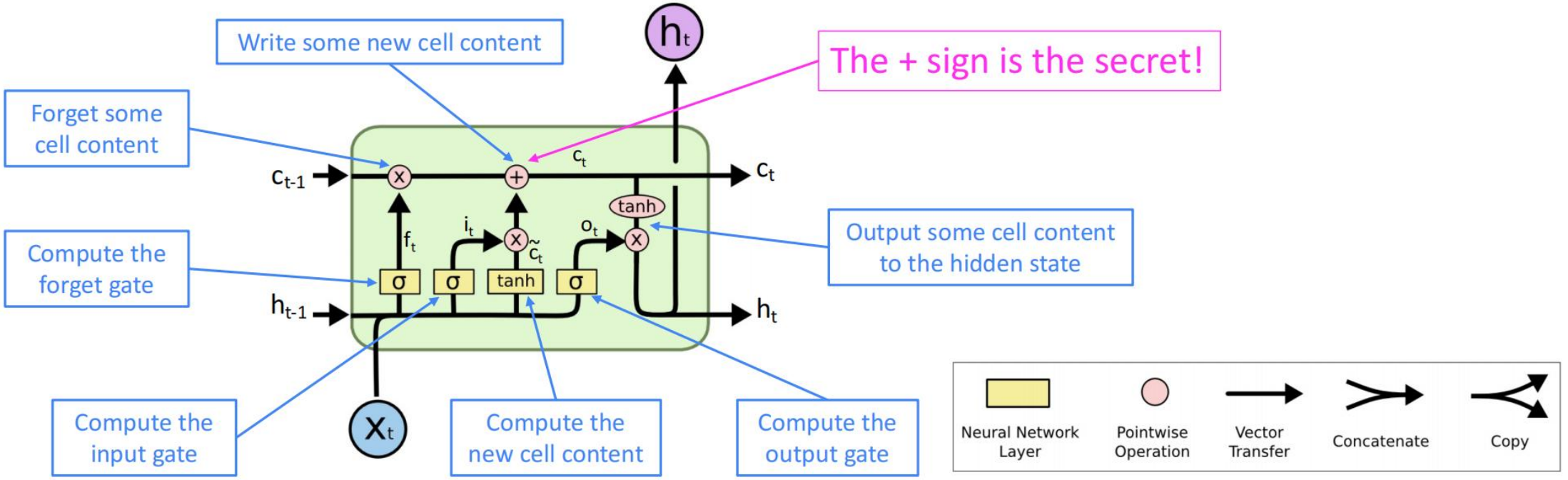
- On step t, there is a hidden state $h^{(t)}$ and a cell state $c^{(t)}$
  - ✓ Both are vectors length n
  - ✓ The cell stores long-term information
  - ✓ The LSTM can read, erase, and write information from the cell

In RNN, hidden neuron is a simple linear sum



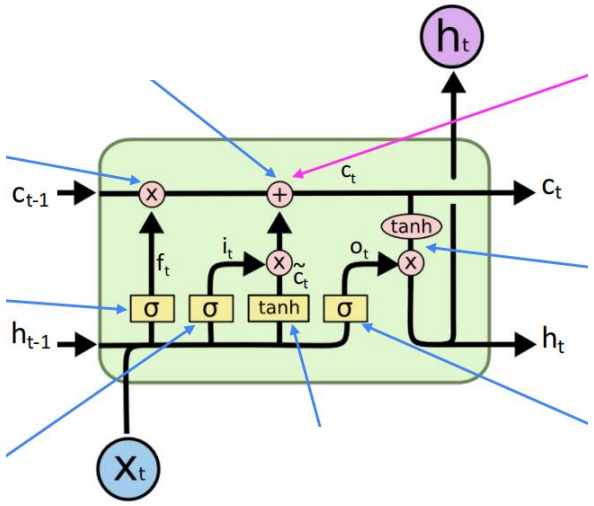
Long Short-Term Memory Networks

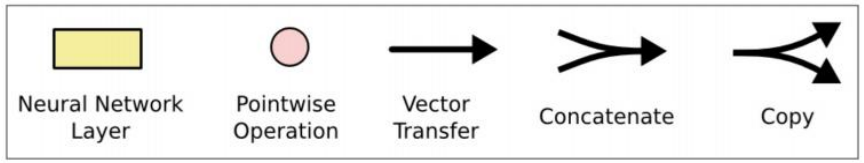
- The selection of which information is erased/written/read is controlled by three gates
  - ✓ The gates are also vectors of length *n*.
  - ✓ On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between.
  - ✓ The gates are dynamic: their value is computed based on the current context.



Long Short-Term Memory Networks

- We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:





Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

Addition

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

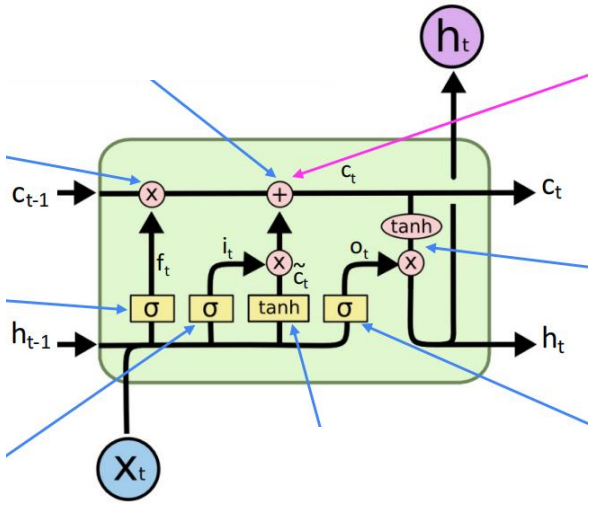
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

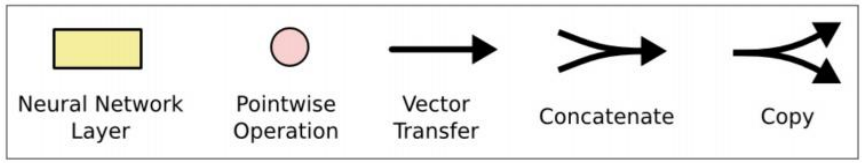
All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$

Long Short-Term Memory Networks

- We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:





Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$f^{(t)} = \sigma)W_f \cdot]h_{t-1}, x_t[+b_f($$

$$i^{(t)} = \sigma)W_i \cdot]h_{t-1}, x_t[+b_i($$

$$o^{(t)} = \sigma)W_o \cdot]h_{t-1}, x_t[+b_o($$

Concatenation

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

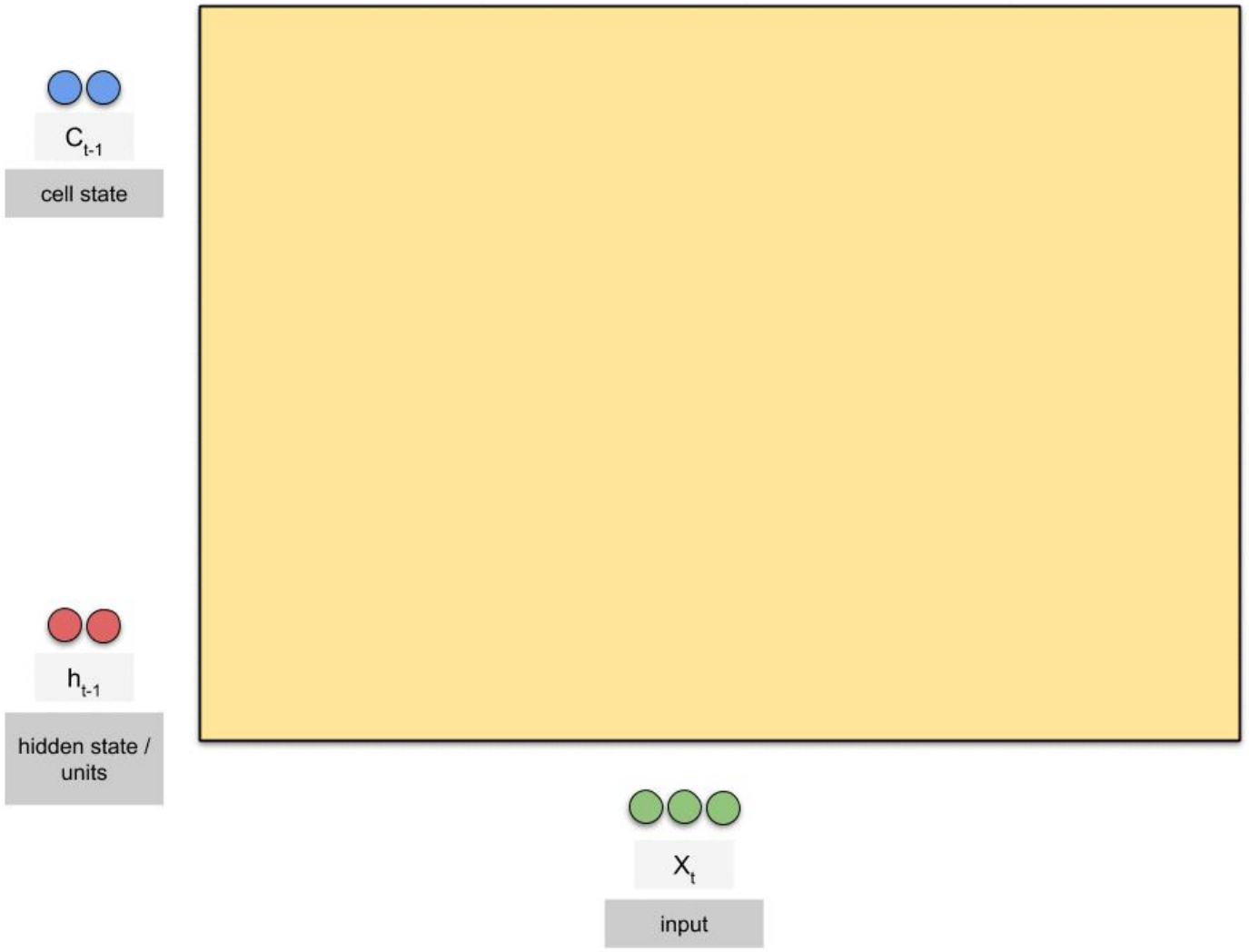
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: ⊙

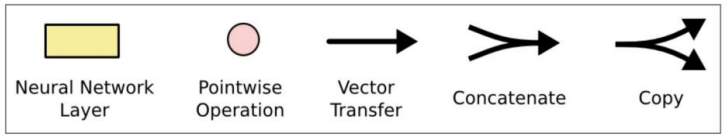
Long Short-Term Memory Networks

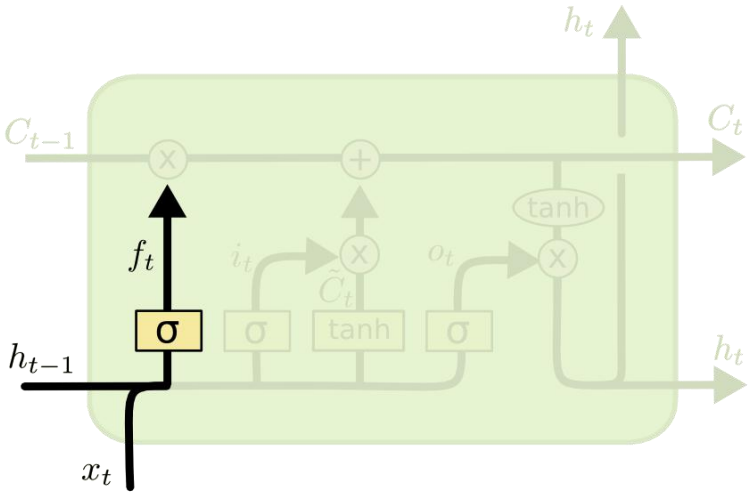


Step-by-step LSTM Walk Through

- **Step 1**: Decide what information to **throw away** from the cell state (memory)
  - ✓ The output of the previous state h_{t-1} and the new information x_t jointly determine what to forget.
  - ✓ h_{t-1} contains selected features from the memory C_{t-1}
  - ✓ **Forget gate** f_t ranges between $[0,\ 1]$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$



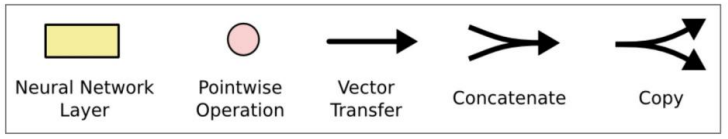


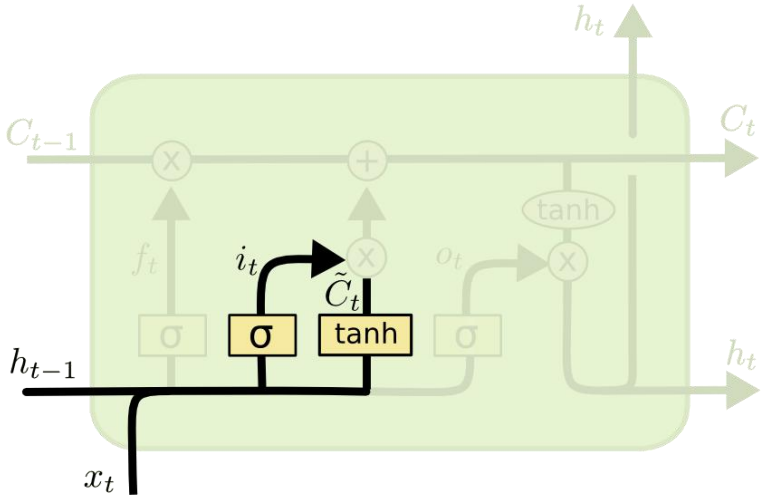
Step-by-step LSTM Walk Through

- **Step 2**: Prepare the updates for the cell state from input
  - ✓ An alternative cell state $\tilde{C}_t$ is created from the new information x_t with the guidance of h_{t-1}.
  - ✓ **Input gate** i_t ranges between [0, 1]

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

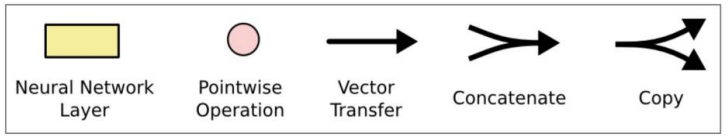


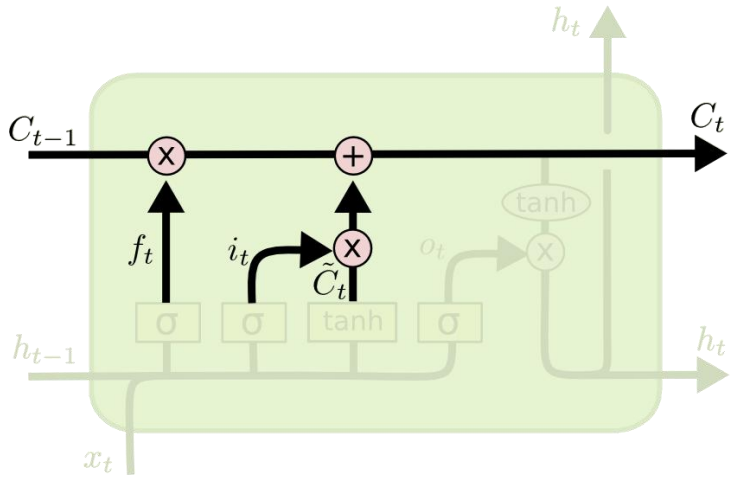


Step-by-step LSTM Walk Through

- **Step 3**: Update the cell state
  - ✓ The new cell state C_t is comprised of information from the past $f_t * C_{t-1}$ and valuable new information $i_t * \tilde{C}_t$
  - ✓ $*$ denotes elementwise multiplication

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



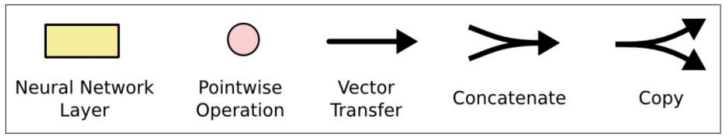


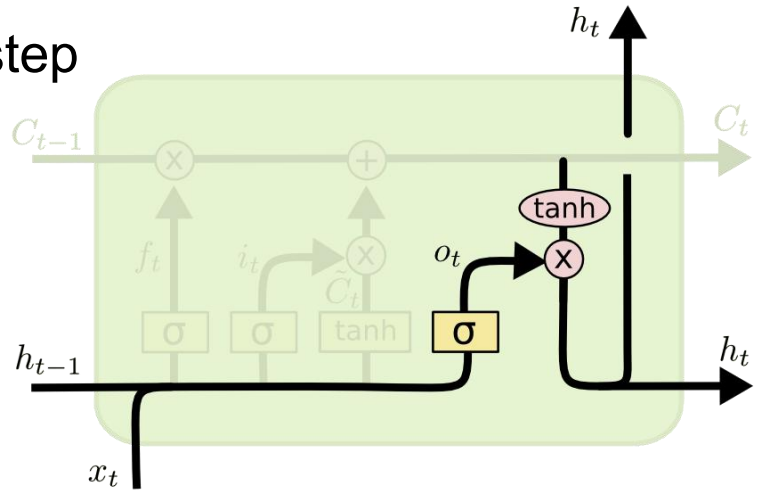
Step-by-step LSTM Walk Through

- **Step 4**: Decide the filtered output from the new cell state
  - ✓ Tanh function filters the new cell state to characterize stored information
    Significant information in $C_t \rightarrow \pm 1$
    Minor details $\rightarrow 0$
  - ✓ **Output gate** o_t ranges between $[0, 1]$
  - ✓ Serves as a control signal for the next time step

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

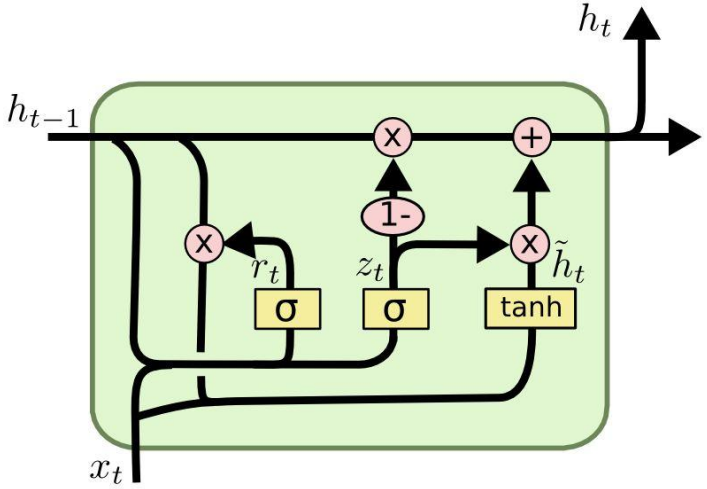
$$h_t = o_t * \tanh(C_t)$$





LSTM variants

- **Gate Recurrent Unit**
  - ✓ It combines the forget and input gates into a single "**update gate**".
  - ✓ It also merges the cell state and hidden state, and makes some other changes.
  - ✓ GRU is simpler than standard LSTM, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

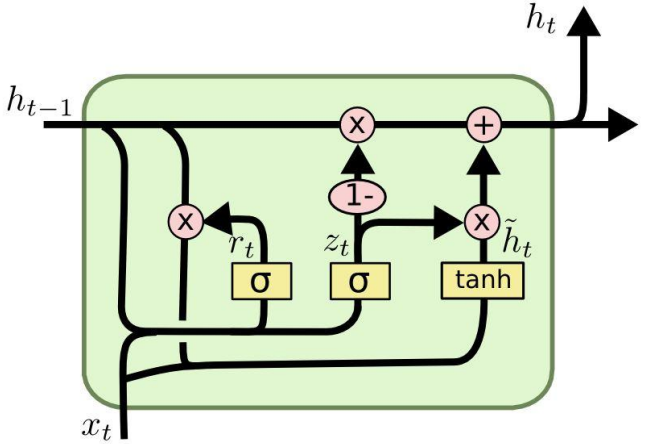
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM variants

- **Gate Recurrent Unit**
  - ✓ **Update gate**: controls the composition of the new state
  - ✓ **Reset gate**: determines how much old information is needed in the alternative state $\tilde{h}_t$
  - ✓ **Alternative state**: contains new information
  - ✓ **New state**: replace selected old information with new information in the new state



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Long Short-Term Memory Networks

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Contents

- Word Representation
- Recurrent Neural Networks
- Long Short-Term Memory Networks
- **Sequence To Sequence**

Pre-Neural Machine Translation

- **Machine Translation (MT)** is the task of translating a sentence *x* from one language (the source language) to a sentence *y* in another language (the target language).
- **Example**:

Source language (English): Machine Translation in natural language processing

Machine Translation Model

Target language (German): Maschinelle Übersetzung in der Verarbeitung natürlicher Sprache

1950s: Early Machine Translation

- Machine Translation research began in the early 1950s.

✓ Russian → English (motivated by the Cold War!)

✓ Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts

1990s-2010s: Statistical Machine Translation

- **Core idea**: Learn a probabilistic model from data
- Suppose we are translating French → English
- We want to find best English sentence y, given French sentence x

$$argmax_y P(y|x)$$

- Use Bayes Rule to break this down into two components to be learnt separately:

$$argmax_y P(x|y)P(y)$$

Translation Model

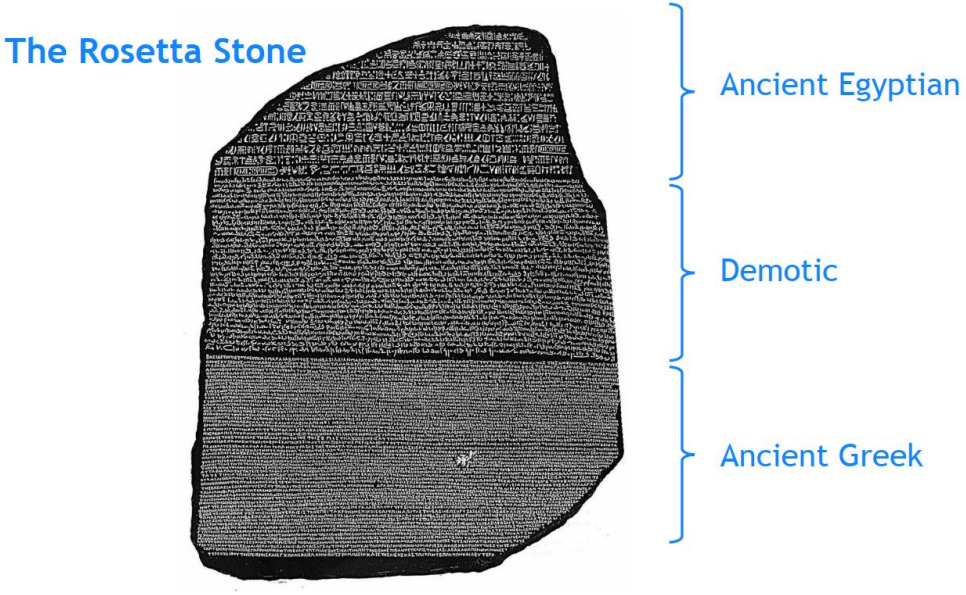
Models how words and phrases should be translated (fidelity). Learnt from parallel data.

Language Model

Models how to write good English (fluency). Learnt from monolingual data.

1990s-2010s: Statistical Machine Translation

- **Question**: How to learn translation model $P(x|y)$?
- First, need large amount of parallel data (e.g. pairs of human-translated French/English sentences)

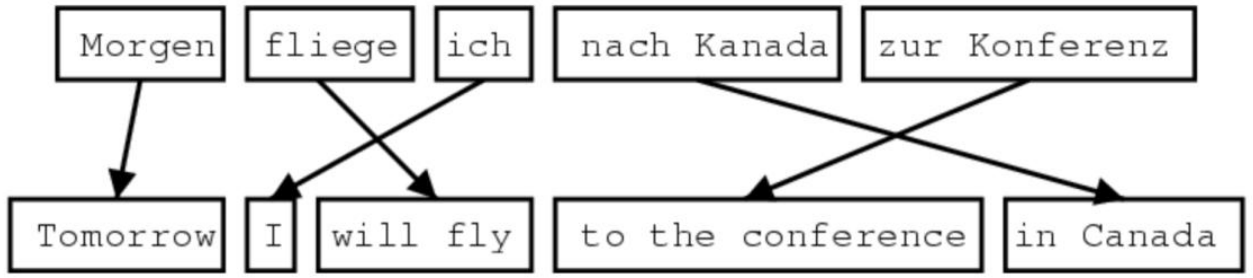


What is Alignment?

- **Question**: How to learn translation model $P(x|y)$ from the parallel corpus?
- Break it down further: Introduce latent a variable into the model:

$$P(x, a|y)$$

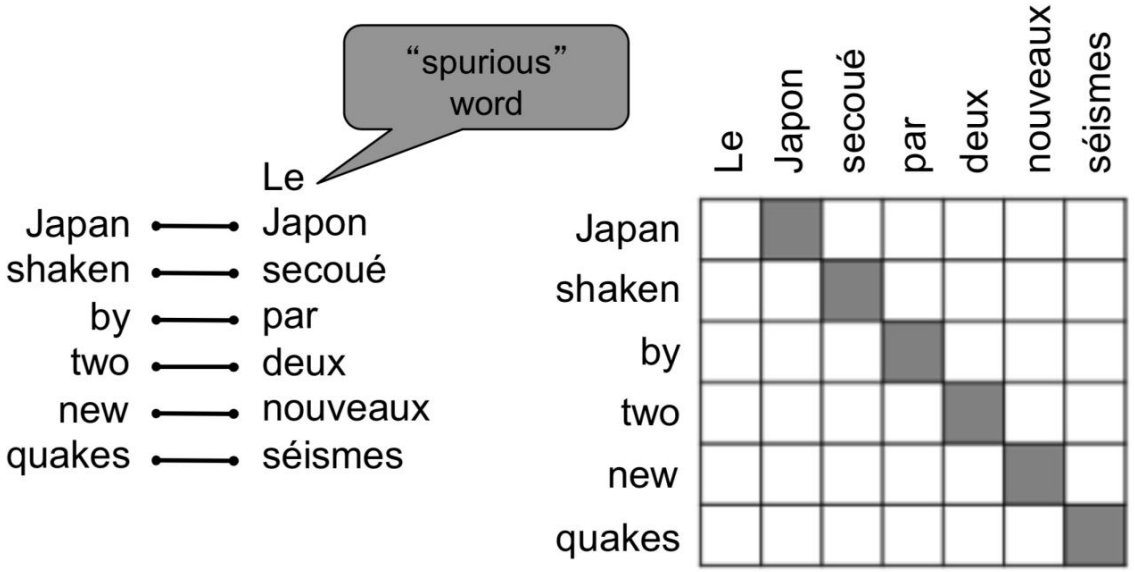
where a is the alignment, i.e. word-level correspondence between source sentence x a



What is Alignment?

Alignment is the correspondence between particular words in the translated sentence pair.

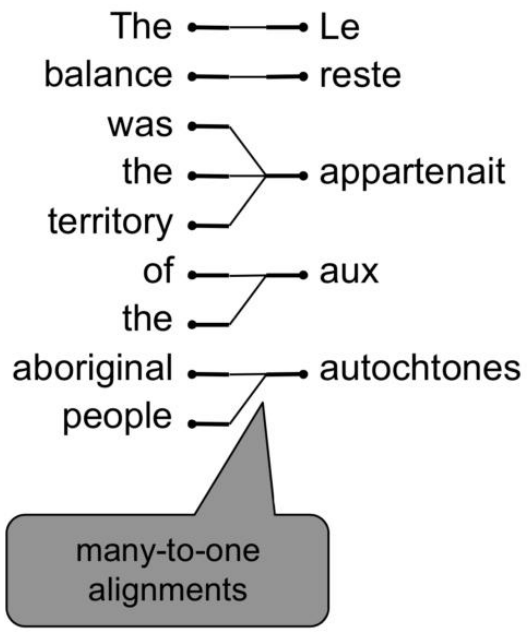
- Typological differences between languages lead to complicated alignments?
- Note: Some words have no counterpart

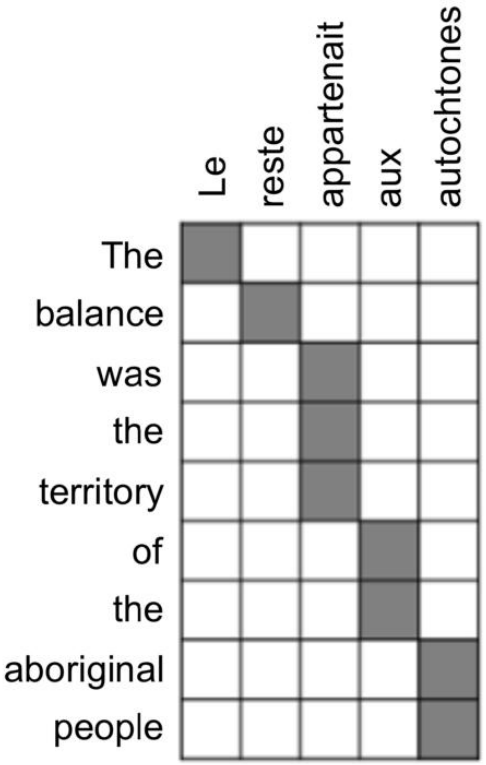


Examples from: "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

Alignment is Complex

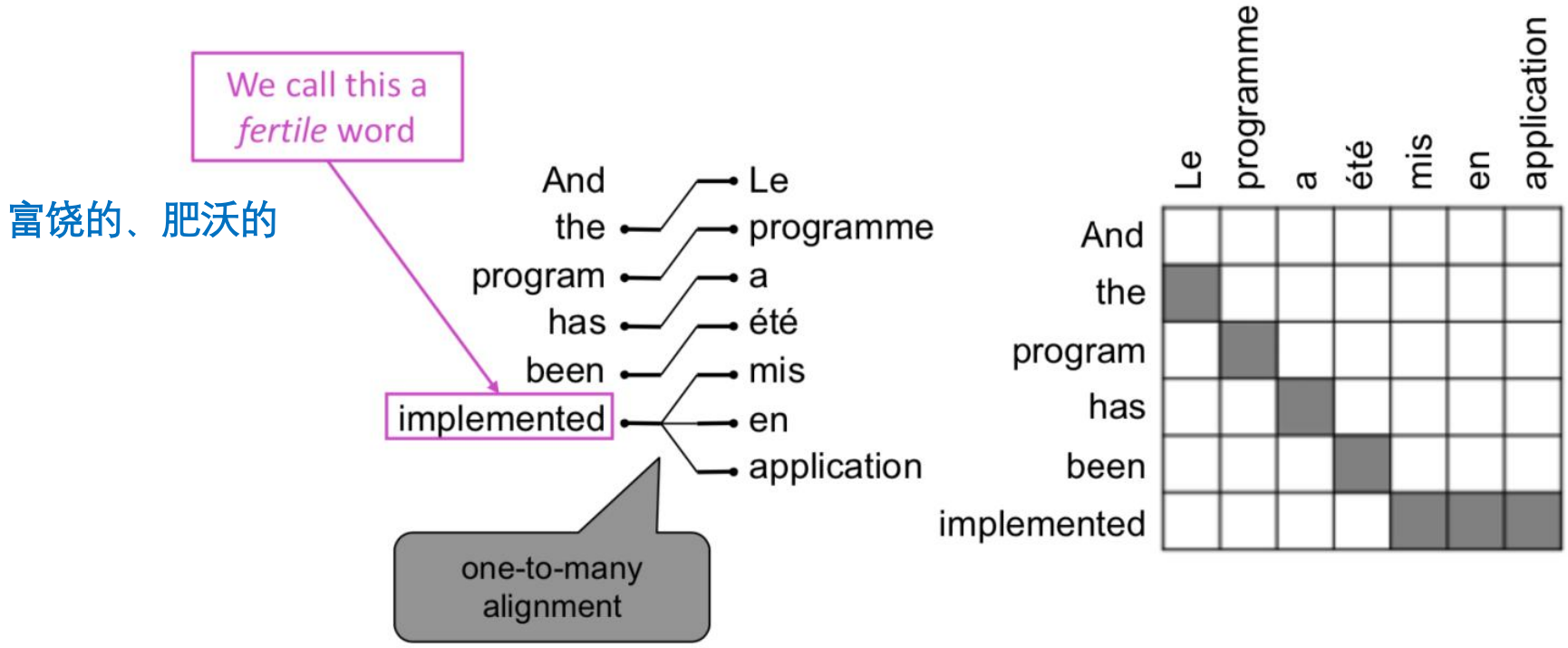
- Alignment can be many-to-one





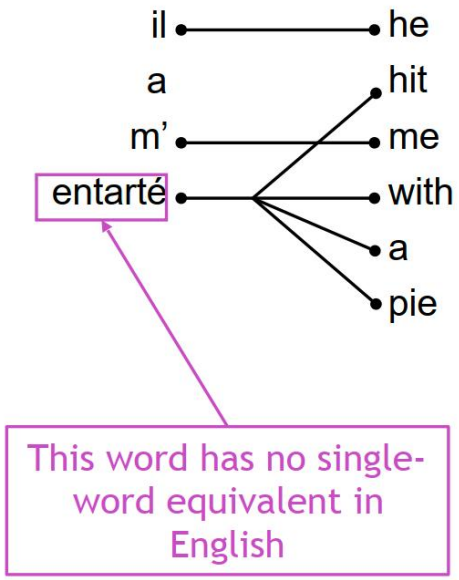
Alignment is Complex

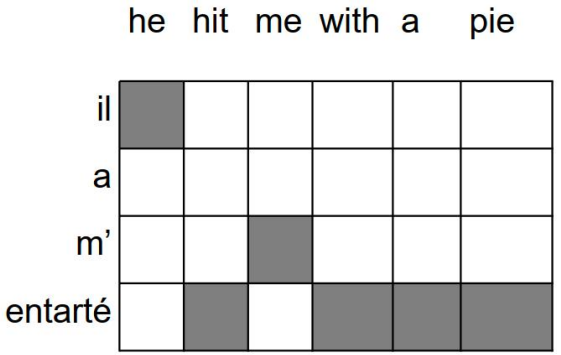
- Alignment can be one-to-many



Alignment is Complex

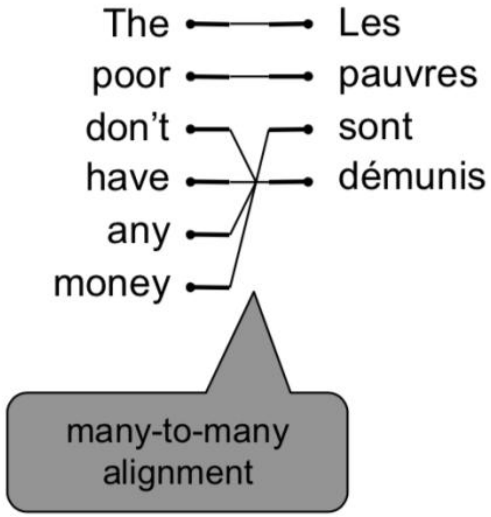
- Some words are very fertile!

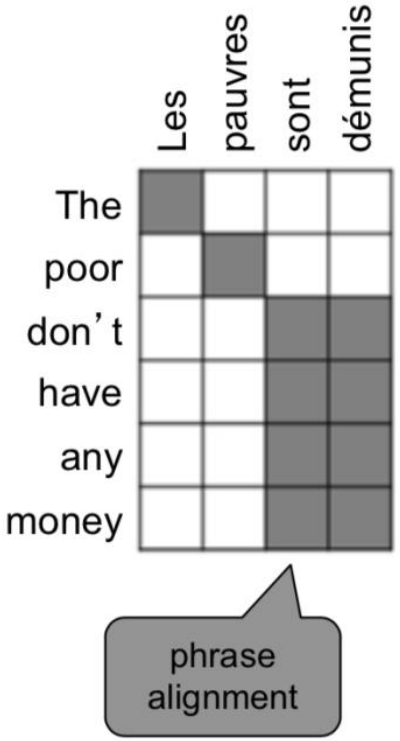




Alignment is Complex

- Alignment can be many-to-many (phrase-level)





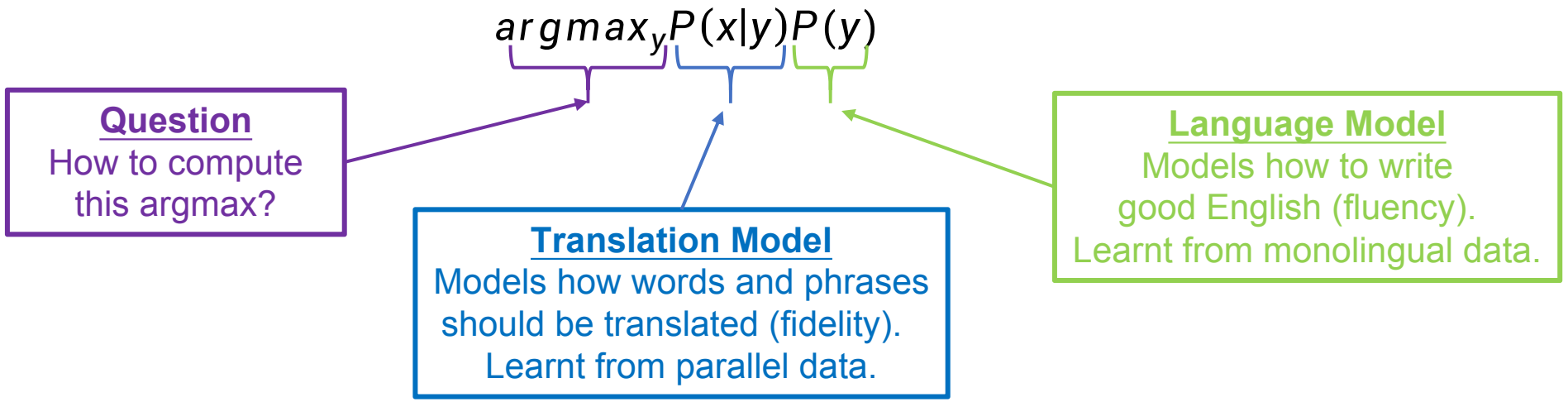
Learning Alignment for SMT

- Not We learn $P(x, a|y)$ as a combination of many factors, including:
  - Probability of particular words aligning (also depends on position in sent)
  - Probability of particular words having particular fertility (number of corresponding words)
  - etc.

- Alignments a are latent variables: They are not explicitly specified in the data!
  - ✓ Require the use of special learning aglos (like Expectation Maximization) for learning the parameters of distributions with latent variables

Decoding for SMT

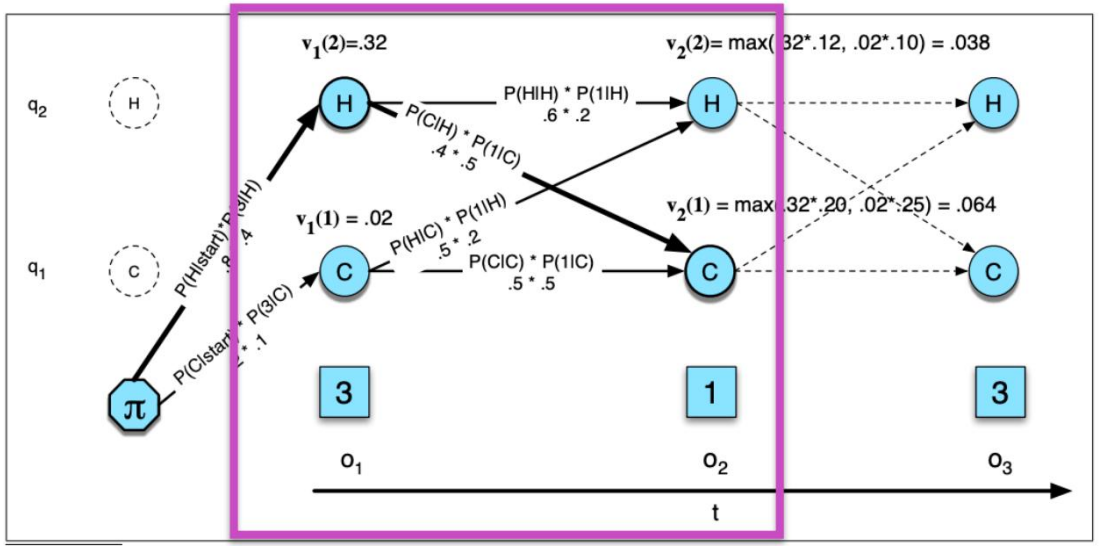
- We could enumerate every possible y and calculate the probability? → Too expensive!
- Answer: Impose strong independence assumptions in model, use dynamic programming for globally optimal solutions (e.g. Viterbi algorithm).
- This process is called decoding



Viterbi: Decoding with Dynamic Programming

- Impose strong independence assumptions in model:

$$P(x, a|y) = \prod_{j=1}^{l_x} p(x_j | f_{a(j)})$$



SMT was a huge research field

The best systems were extremely complex

- Hundreds of important details we have not mentioned here
- Systems had many separately-designed subcomponents
- Lots of feature engineering
  - Need to design features to capture particular language phenomena
- Require compiling and maintaining extra resources
  - Like tables of equivalent phrases
- Lots of human effort to maintain
  - Repeated effort for each language pair!

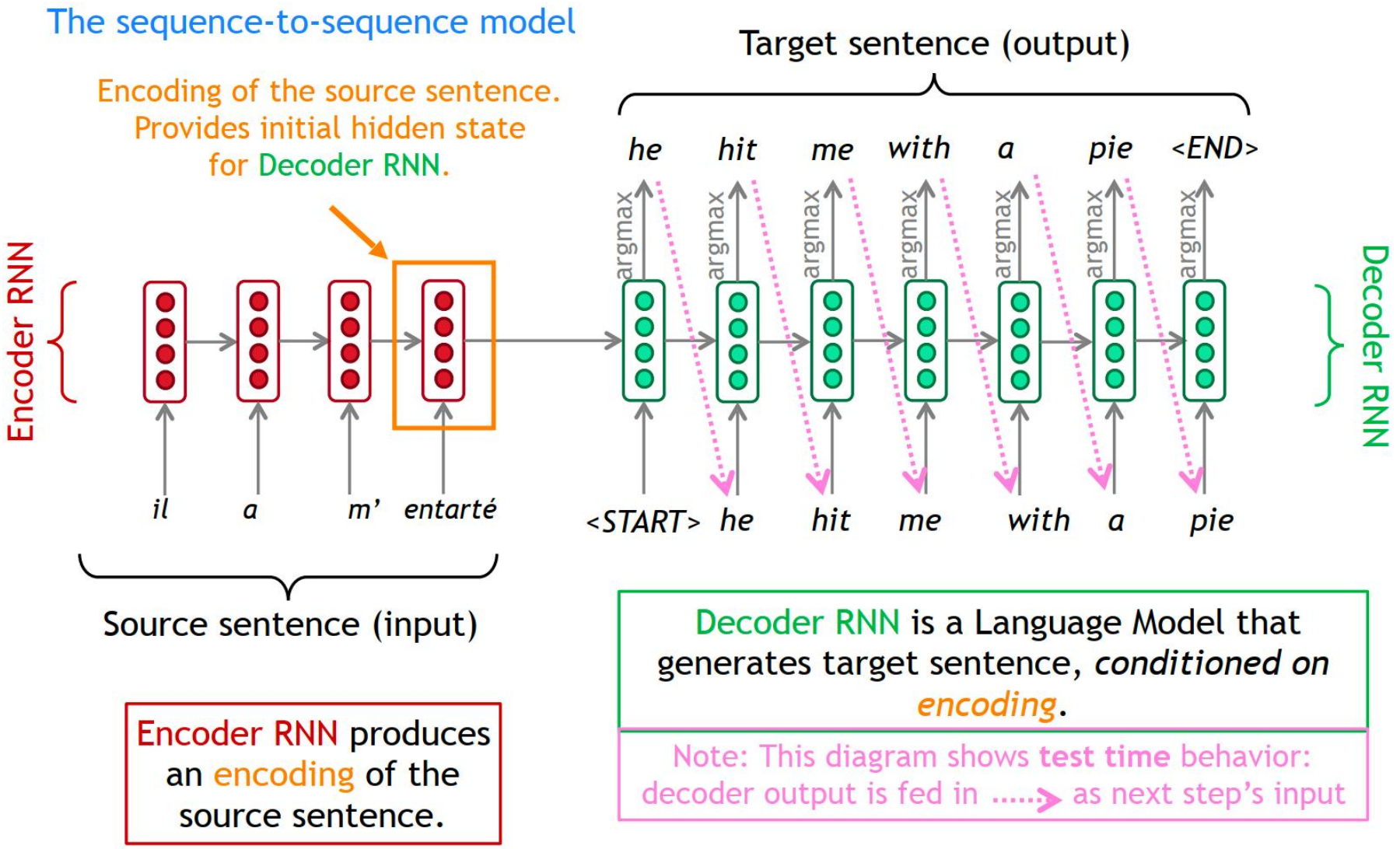
Neural Machine Translation (NMT)

What is Neural Machine Translation?

- Neural Machine Translation (NMT) is a way to do Machine Translation with a single neural network

- The neural network architecture is called **sequence-to-sequence** (aka **seq2seq**) and it involves two RNNs

Sequence To Sequence



Sequence-to-Sequence is Versatile!

- Sequence-to-sequence is useful for more than just MT
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Neural Machine Translation (NMT)

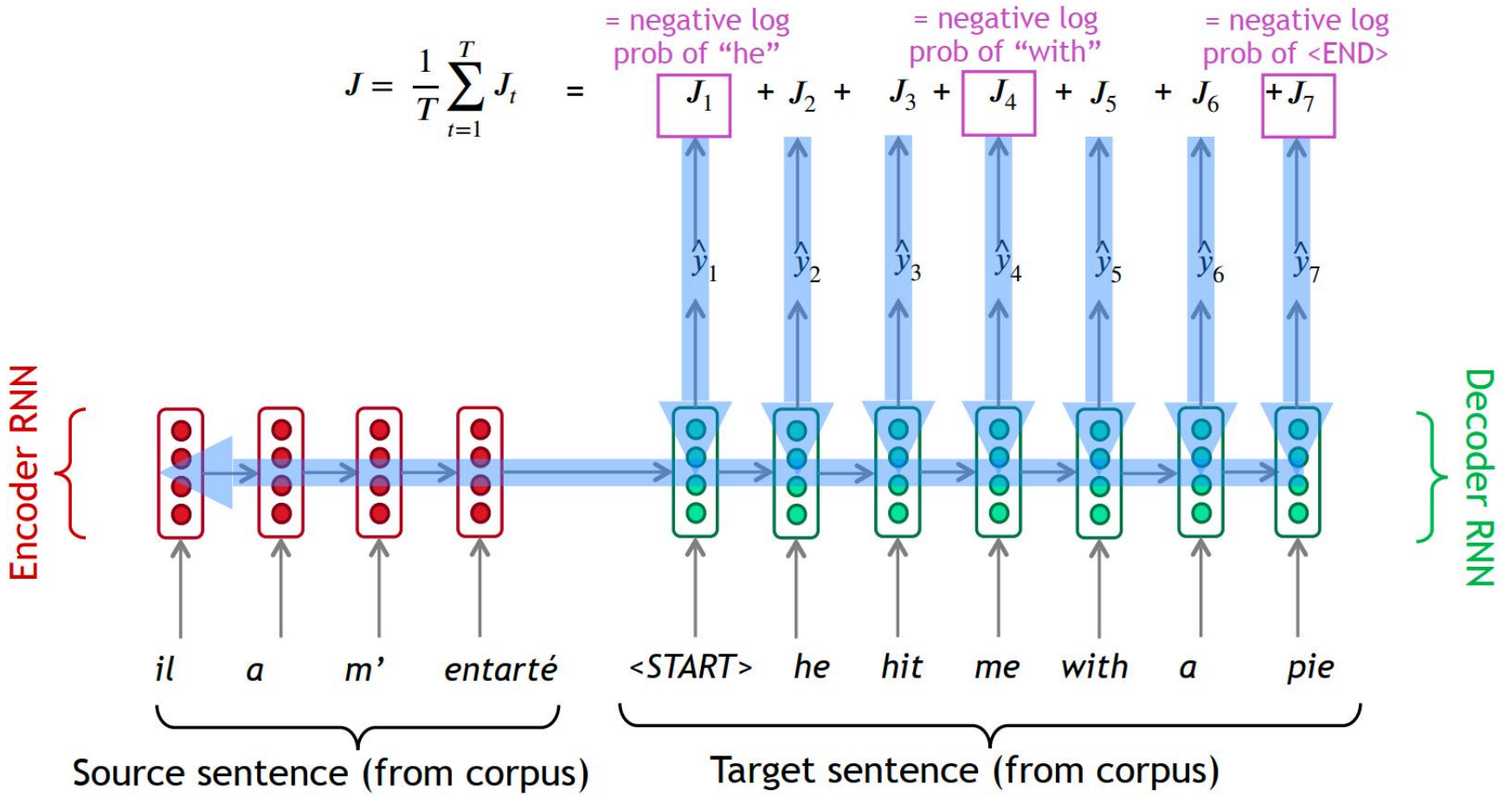
- The sequence-to-sequence model is an example of a Conditional Language Model.
  - ✓ **Language Model** because the decoder is predicting the next word of the target sentence y
  - ✓ **Conditional** because its predictions are also conditioned on the source sentence x

- NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x)\ldots P(\underbrace{y_T|y_1, \ldots, y_{T-1}, x}_{\text{Probability of next target word, given target words so far and source sentence } x})$$

- **Question**: How to train a NMT system?
- **Answer**: Get a big parallel corpus…

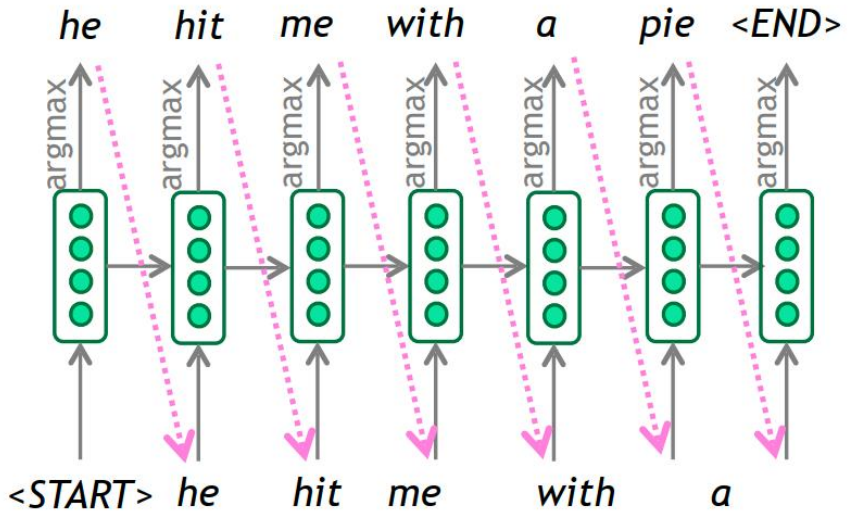
Training a Neural Machine Translation System



Seq2seq is optimized as a **single system**. Backpropagation operates "end-to-end".

Greedy Decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder.



- This is greedy decoding (take most probable word on each step)
- Problems with this method?

Problems with Greedy Decoding

- Greedy decoding has no way to undo decisions!
  - Input: il a m′ entarté (he hit me with a pie)
  - → he _____
  - → he hit _____
  - → he hit a _____ (whoops! no going back now…)
- How to fix this?

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x)...P(y_T|y_1, ..., y_{T-1}, x)$$

$$= \prod_{t=1}^{T} P(y_t|y_1, ..., y_{t-1}, x)$$

- We could try computing all possible sequences y
  - ✓ This means that on each step t of the decoder, we are tracking V^t possible partial translations, where V is vocab size
  - ✓ This $O(V^T)$ complexity is far too expensive!

Beam Search Decoding

- **Core idea**: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - ✓ k is the beam size (in practice around 5 to 10)
- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$score(y_1, \ldots, y_t) = logP_{LM}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$

  - ✓ Scores are all negative, and higher score is better
  - ✓ We search for high-scoring hypotheses, tracking top k on each step

Beam Search Decoding

- **Core idea**: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - ✓ k is the beam size (in practice around 5 to 10)
- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$score(y_1, \ldots, y_t) = logP_{LM}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$

- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Beam Search Decoding: Example

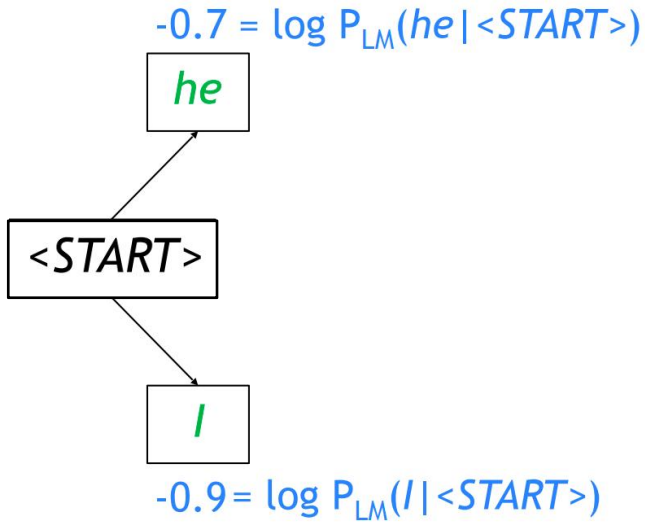
- Beam size = k = 2. Blue numbers $score(y_1, ..., y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$

<START>

Calculate prob dist of next word

Beam Search Decoding: Example

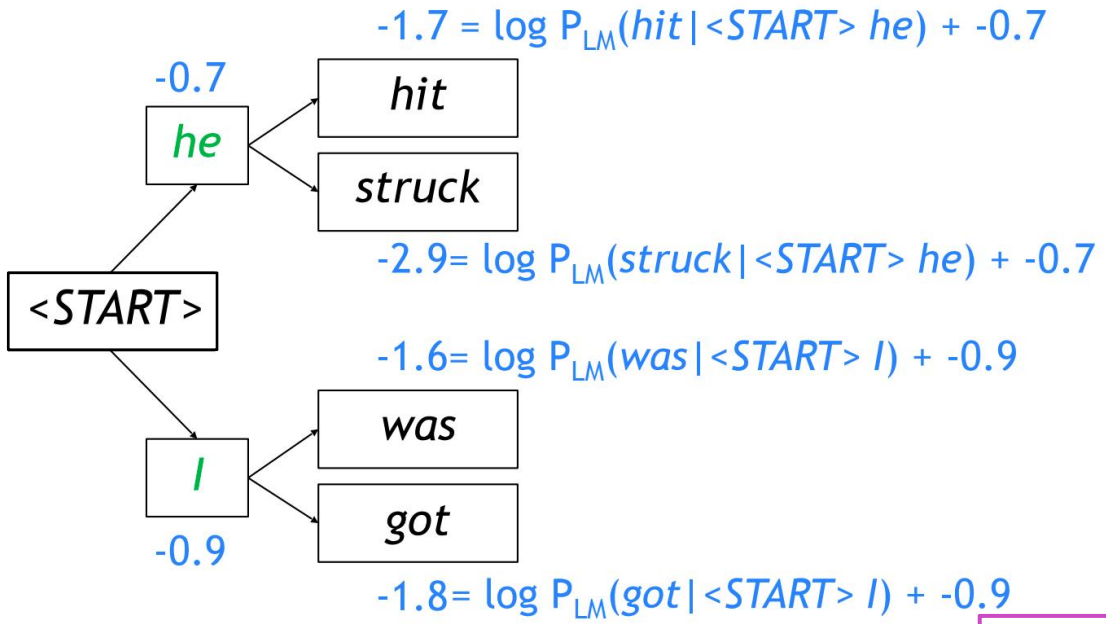
- Beam size = k = 2. Blue numbers $score(y_1, ..., y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$



Take top *k* words and compute scores

Beam Search Decoding: Example

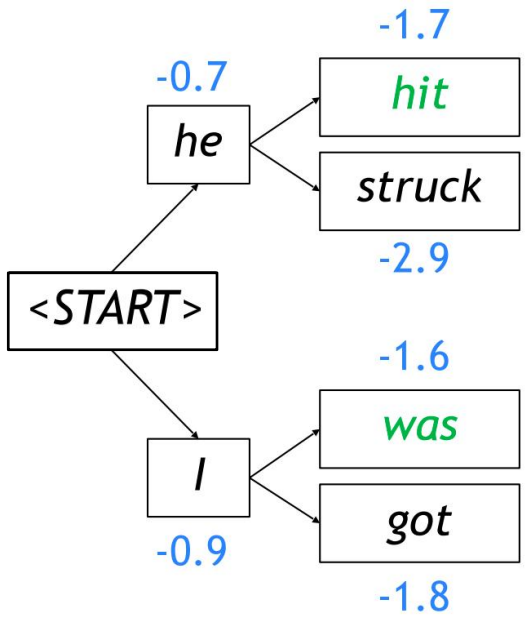
- Beam size = k = 2. Blue numbers $score(y_1, ..., y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam Search Decoding: Example

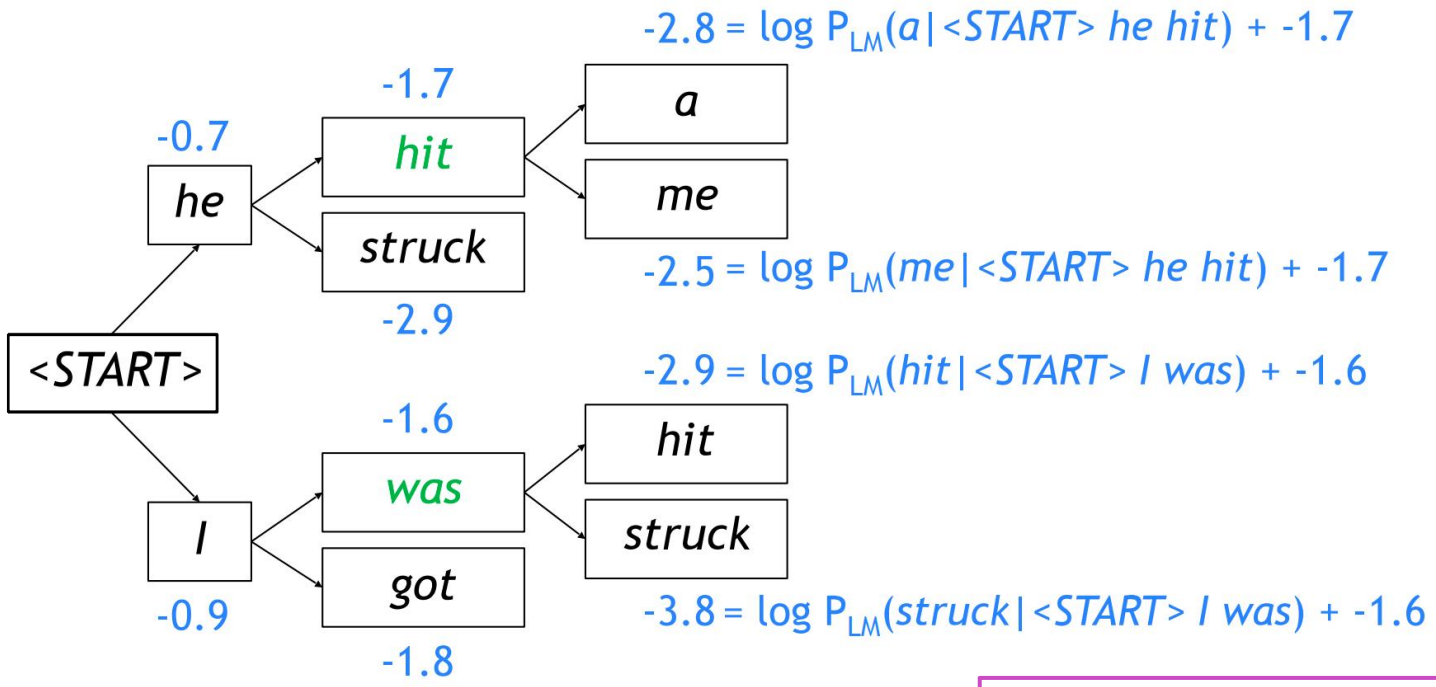
- Beam size = k = 2. Blue numbers $score(y_1, ..., y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$



Of these k^2 hypotheses, just keep k with highest scores

Beam Search Decoding: Example

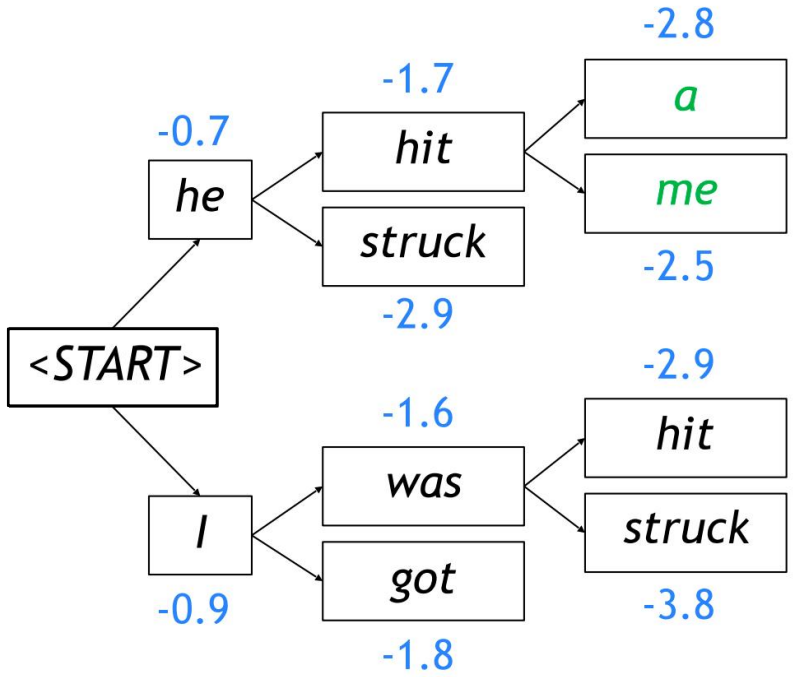
- Beam size = k = 2. Blue numbers $score(y_1, ..., y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

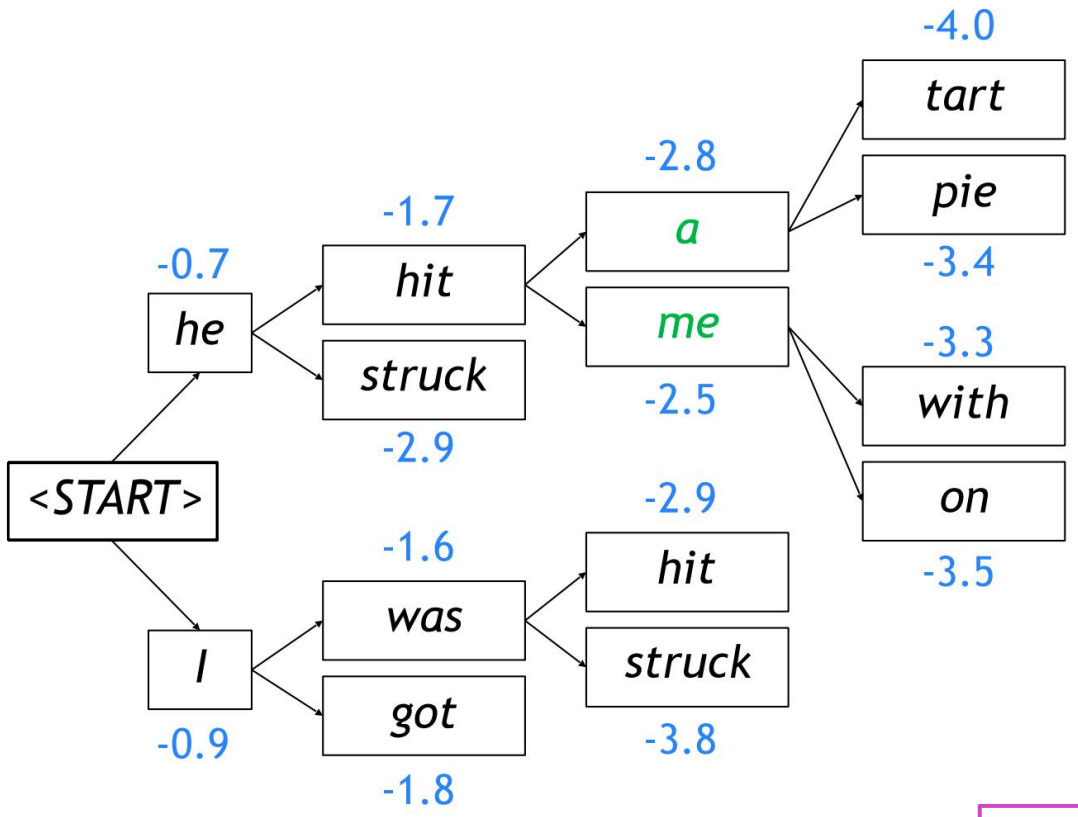
Beam Search Decoding: Example

- Beam size = k = 2. Blue numbers $score(y_1, ..., y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$



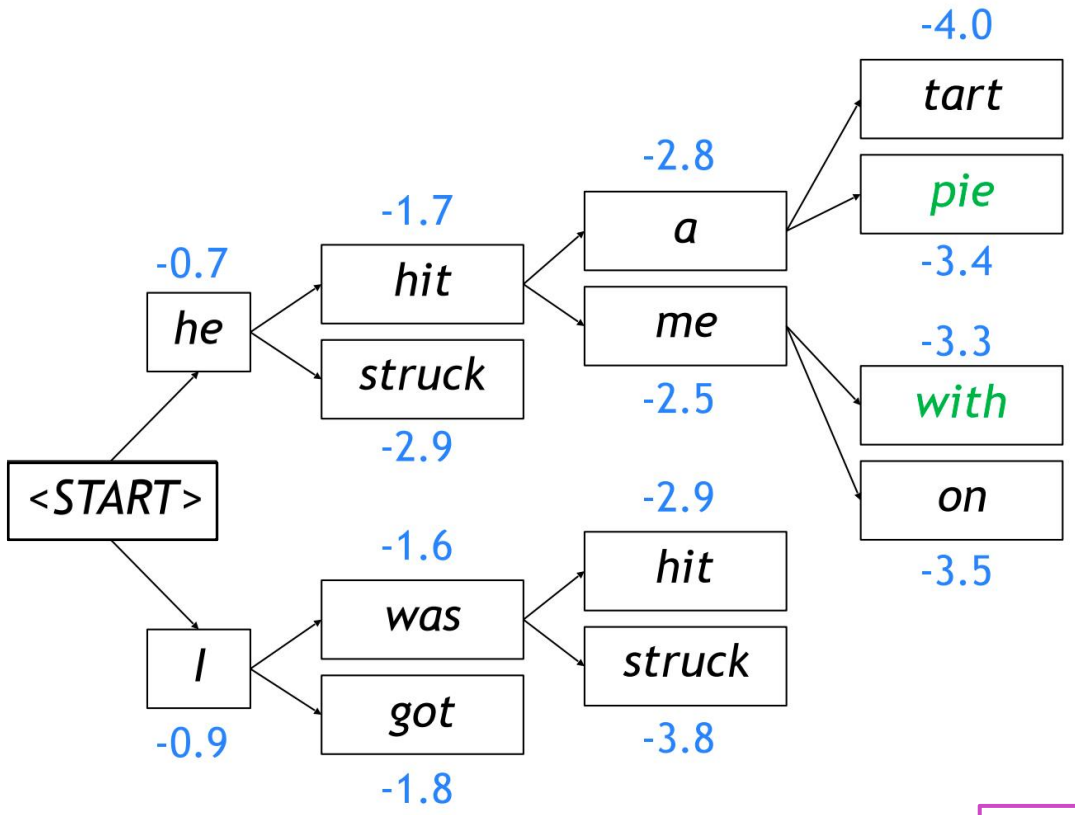
Of these k^2 hypotheses, just keep k with highest scores

Beam Search Decoding: Example



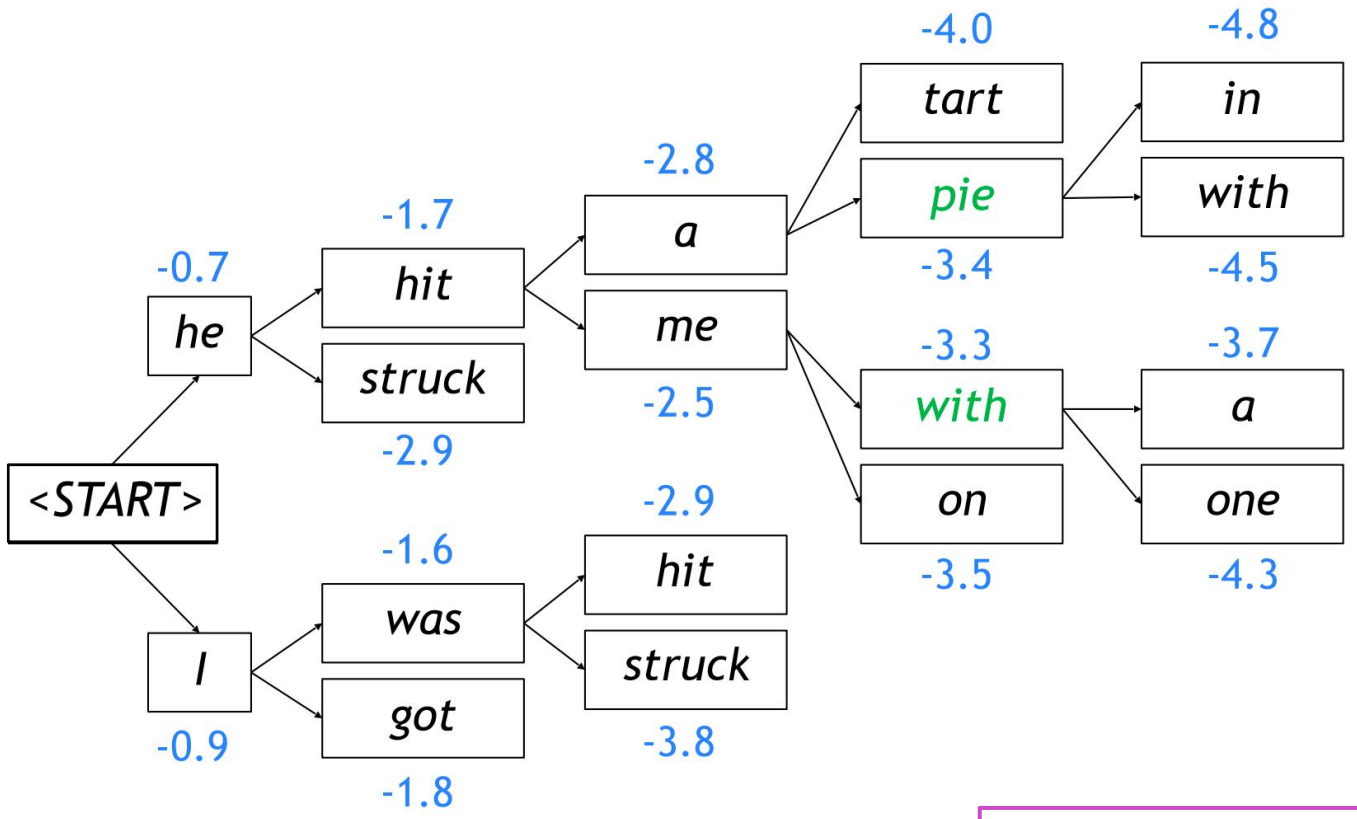
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam Search Decoding: Example



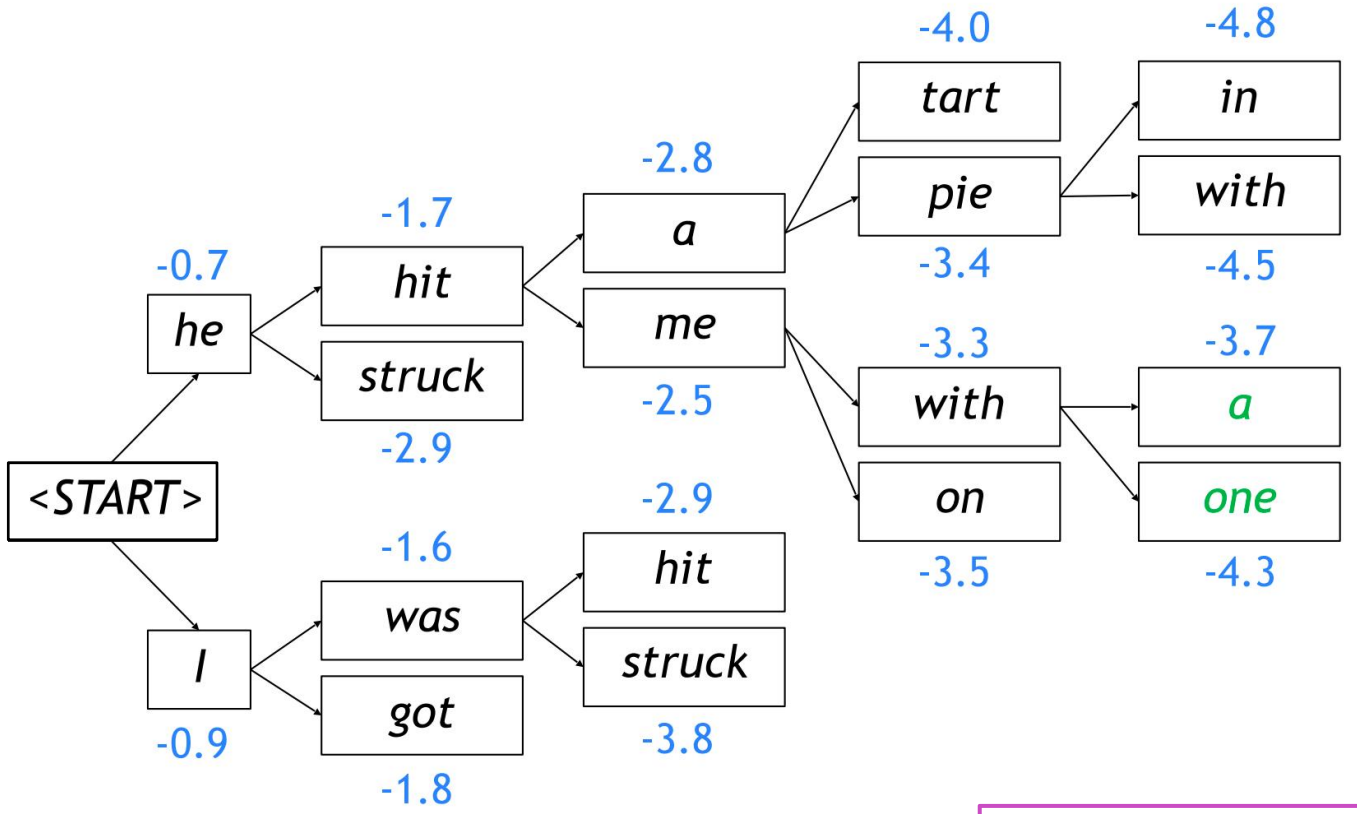
Of these k^2 hypotheses, just keep k with highest scores

Beam Search Decoding: Example



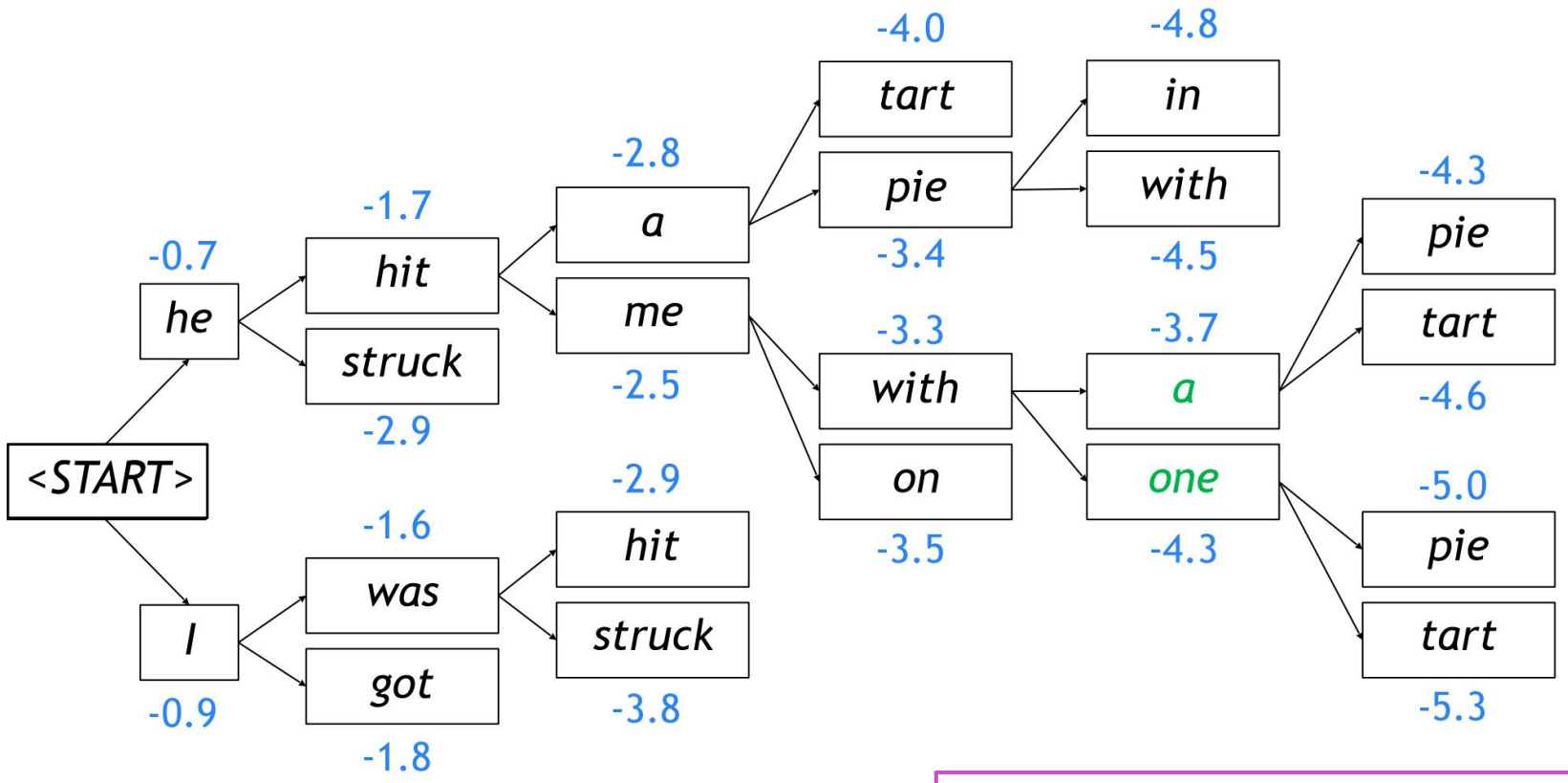
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam Search Decoding: Example



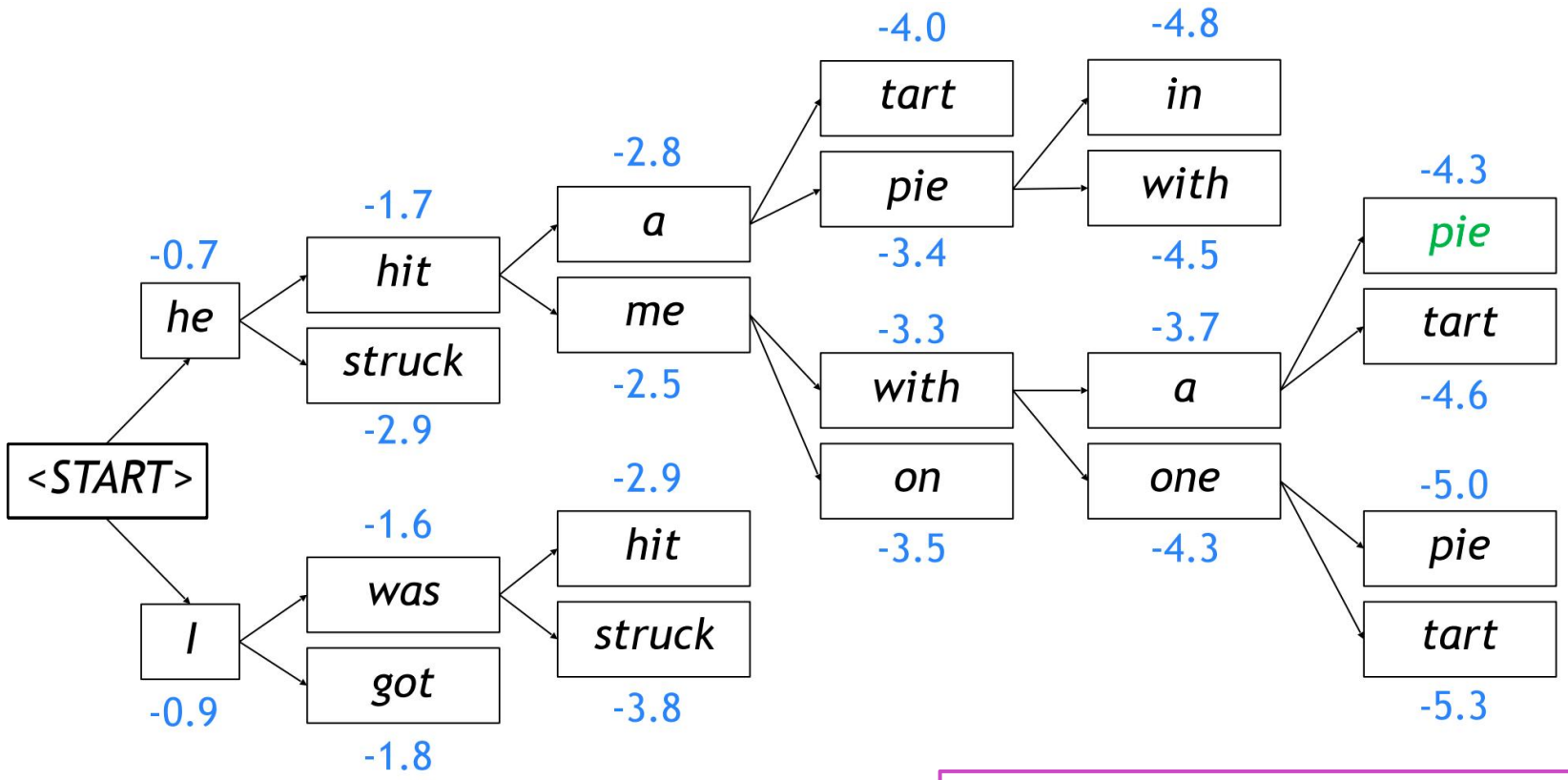
Of these k^2 hypotheses, just keep k with highest scores

Beam Search Decoding: Example



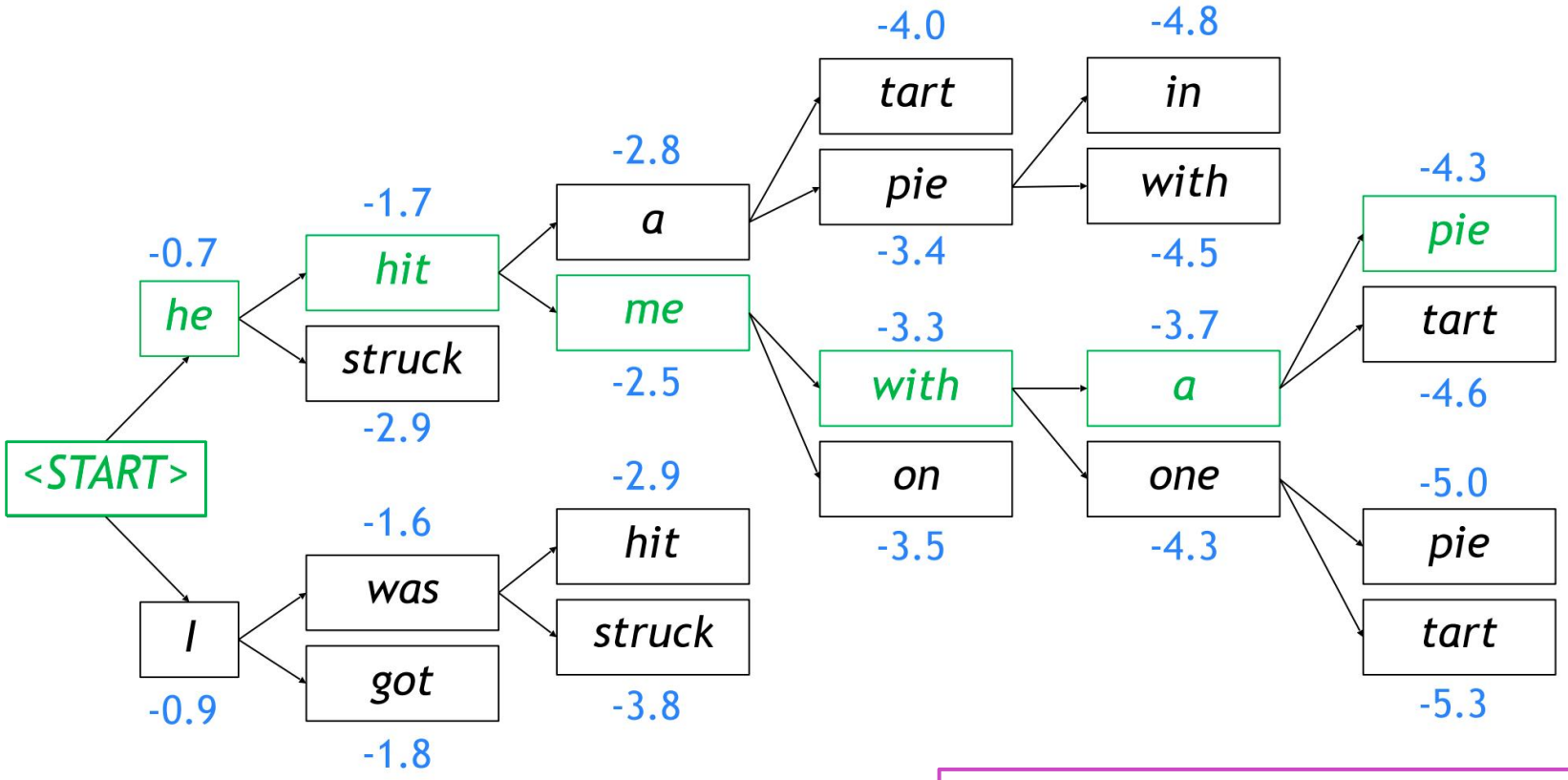
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam Search Decoding: Example



This is the top-scoring hypothesis!

Beam Search Decoding: Example



Backtrack to obtain the full hypothesis

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces a <END> token
  - For example: <START> he hit me with a pie <END>
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

Beam Search Decoding: Finishing up

- We have our list of completed hypotheses.
- How to select top one with highest score?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$score(y_1, \ldots, y_t) = logP_{LM}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$

- **Problem with this**: longer hypotheses have lower scores
- **Fix**: Normalize by length. Use this to select top one instead:

$$\frac{1}{t}\sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$

Advantages of NMT

Compared to SMT, NMT has many advantages:

- Better performance
  - ✓ More fluent
  - ✓ Better use of context
  - ✓ Better use of phrase similarities
- A single neural network to be optimized end-to-end
  - ✓ No subcomponents to be individually optimized
- Requires much less human engineering effort
  - ✓ No feature engineering
  - ✓ Same method for all language pairs

Disadvantages of NMT?

Compared to SMT:

- NMT is less interpretable
  - Hard to debug
- NMT is difficult to control
  - For example, can't easily specify rules or guidelines for translation
  - Safety concerns!

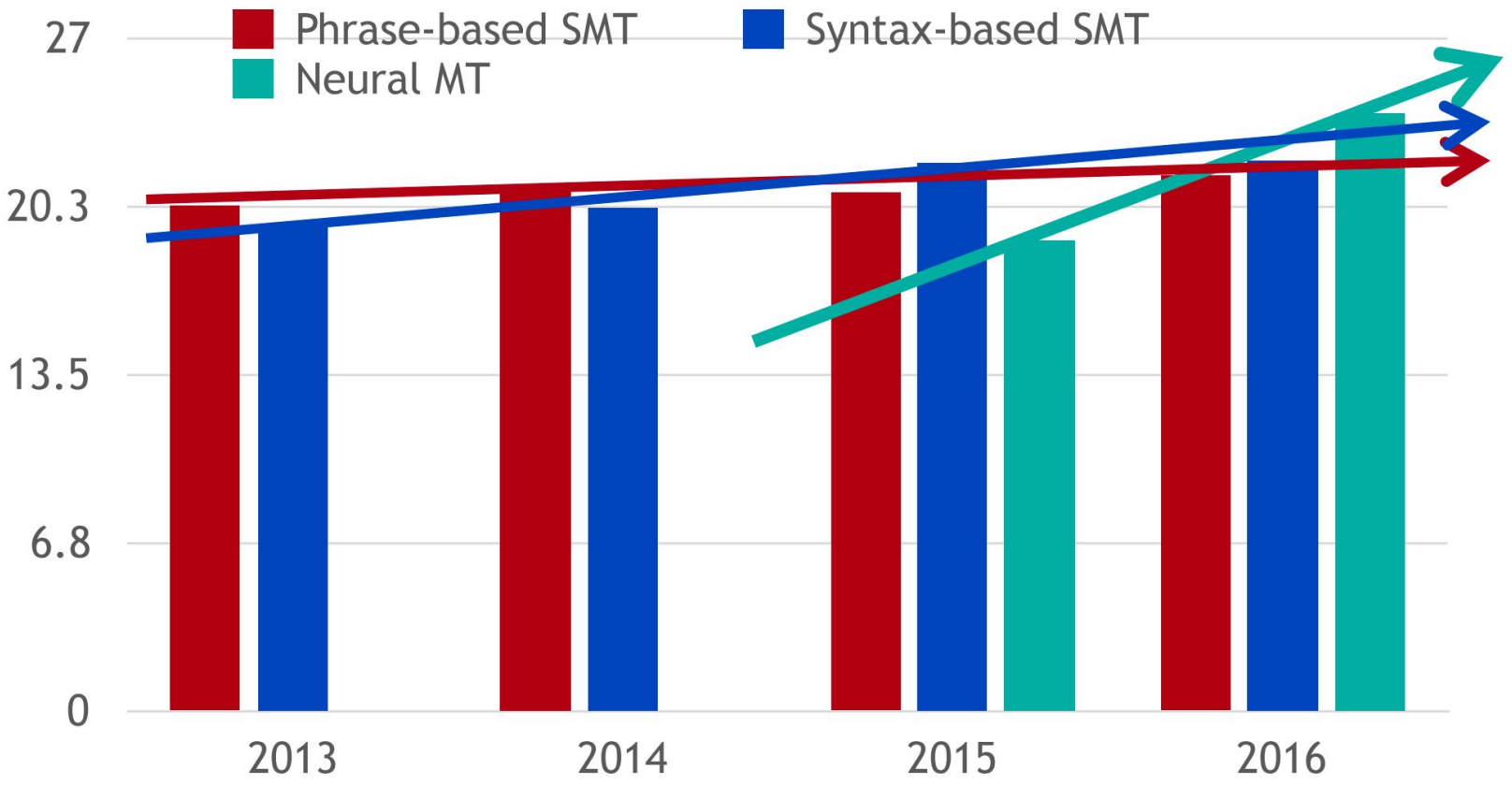
How do we evaluate Machine Translation?

BLEU (Bilingual Evaluation Understudy)

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - ✓ n-gram precision (usually for 1, 2, 3 and 4-grams)
  - ✓ Plus a penalty for too-short system translations

- BLEU is useful but imperfect
  - ✓ There are many valid ways to translate a sentence
  - ✓ So a good translation can get a poor BLEU score because it has low n-gram overlap with the human translation

MT Progress over Time



NMT: The Biggest Success Story of NLP DL

Neural Machine Translation went from a fringe research activity in **2014** to the leading standard method in **2016**

- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT

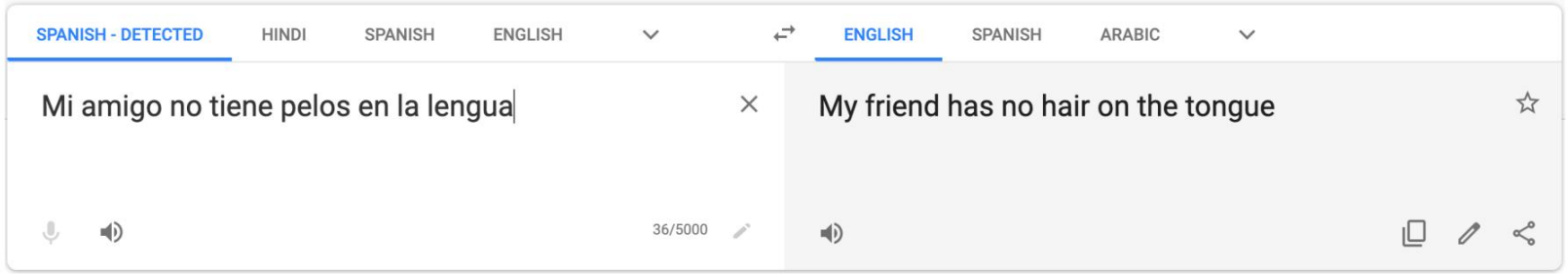
- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

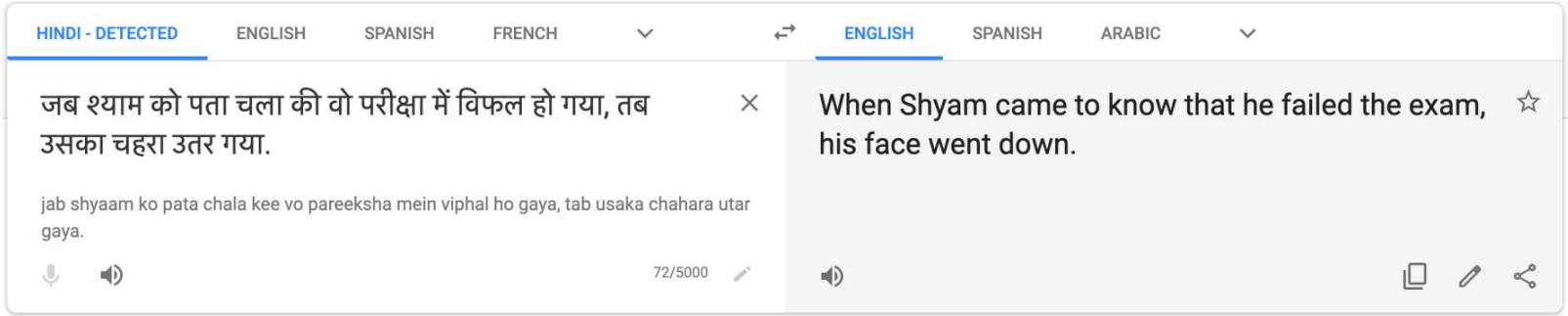
So is Machine Translation Solved?

- Nope!
- Many difficulties remain:
  - ✓ Out-of-vocabulary words
  - ✓ Domain mismatch between train and test data
  - ✓ Maintaining context over longer text
  - ✓ Low-resource language pairs

So is Machine Translation Solved?

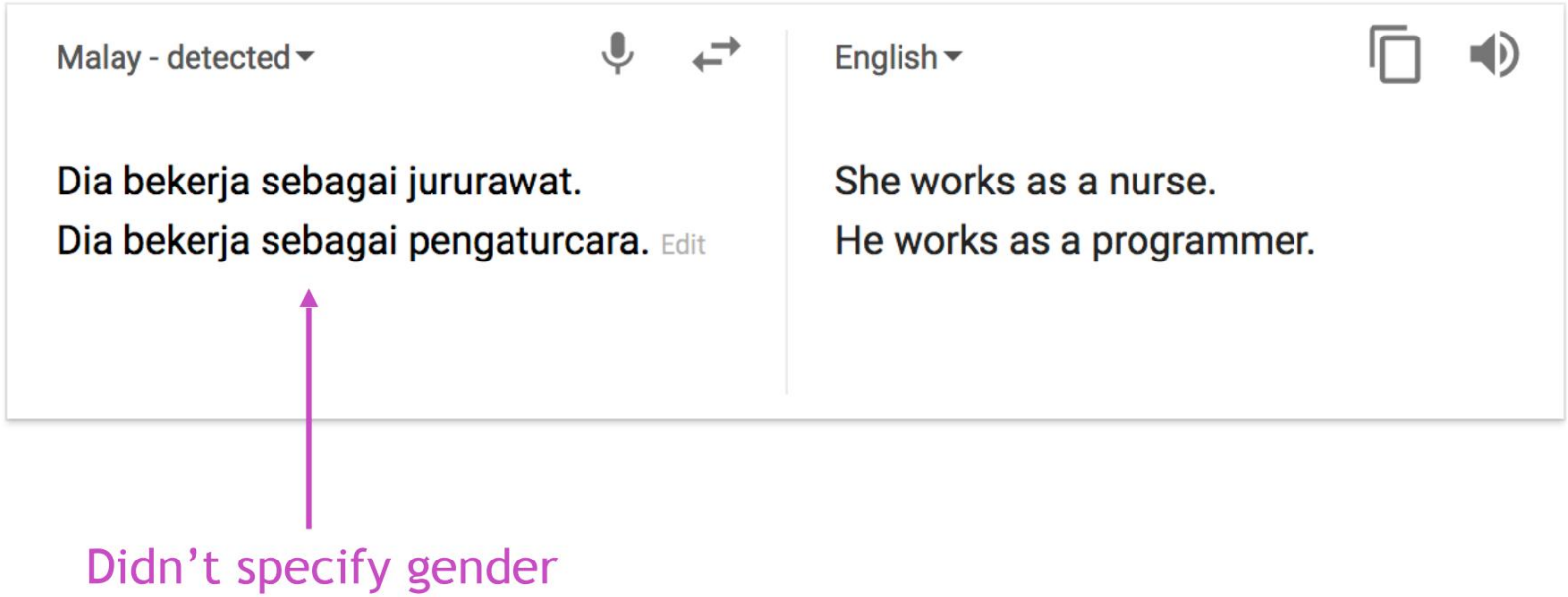
- Using common sense is still hard
- Idioms are difficult to translate





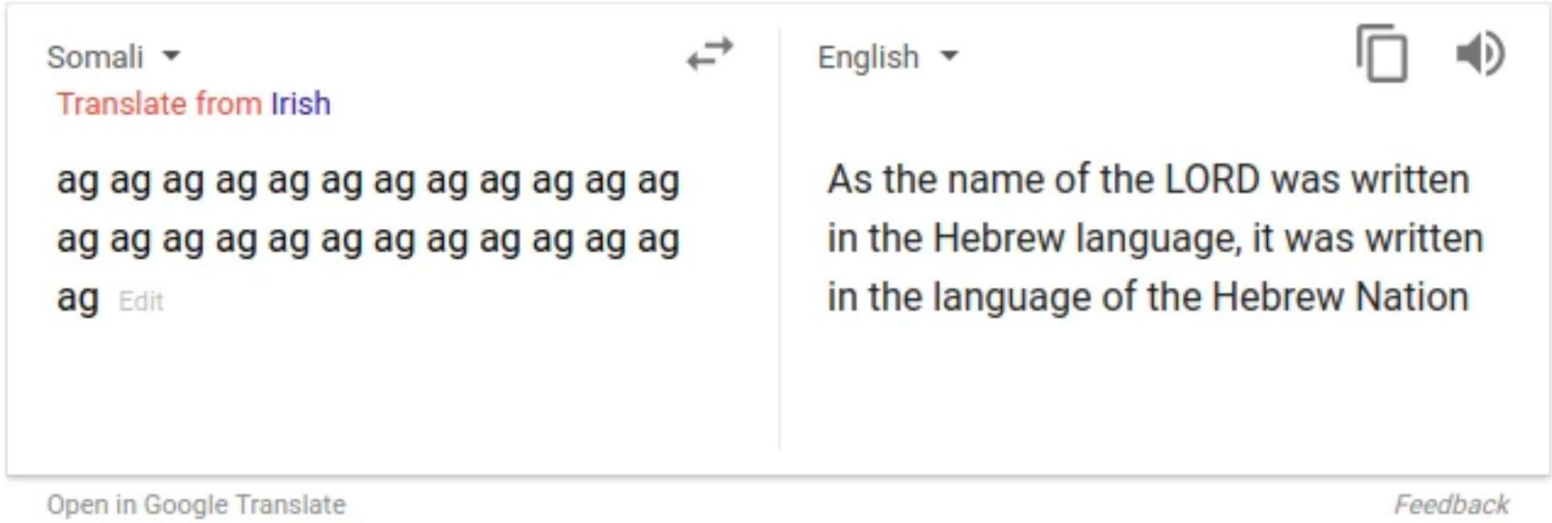
So is Machine Translation Solved?

- NMT picks up biases in training data



So is Machine Translation Solved?

- Uninterpretable systems do strange things



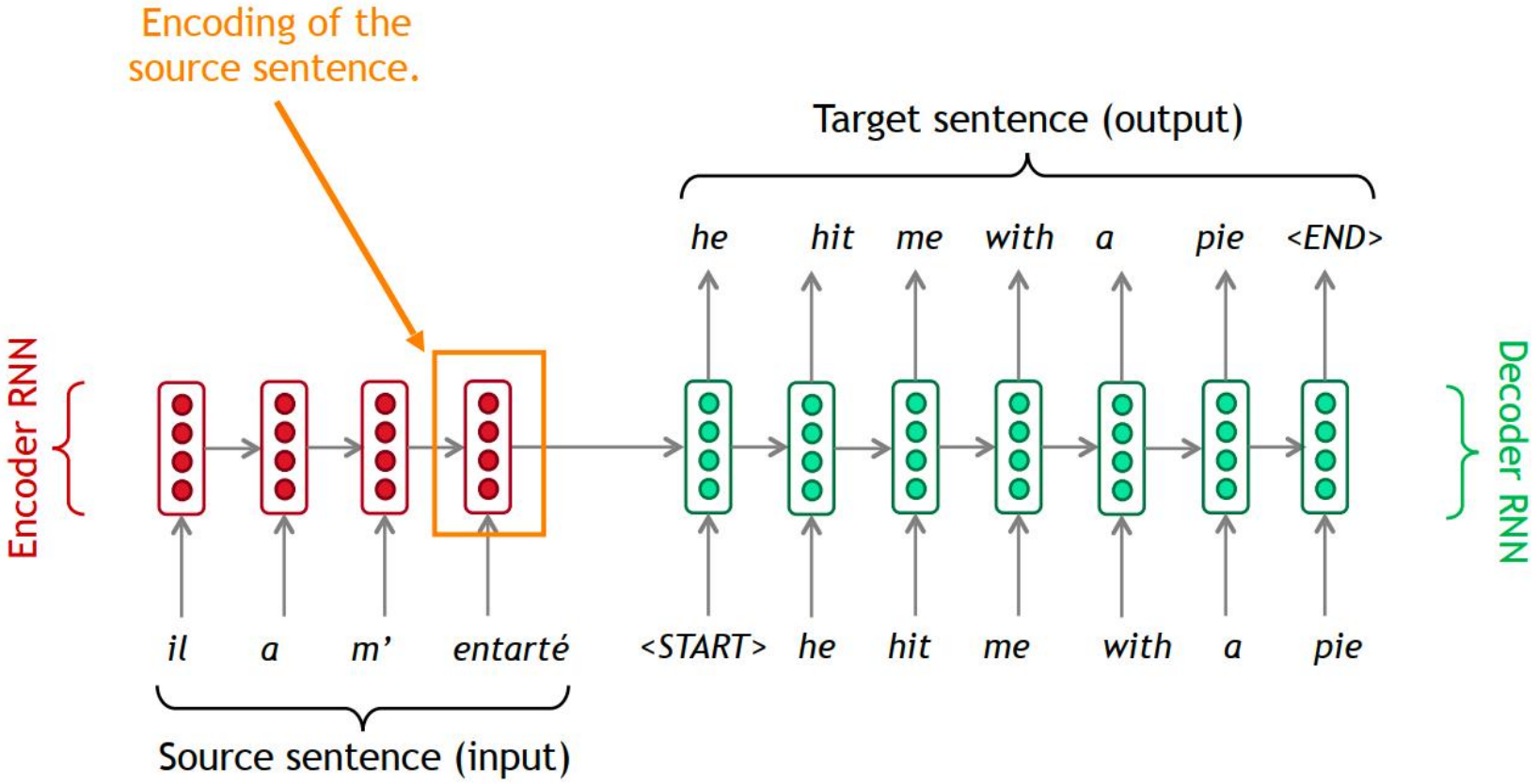
NMT Research Continues

NMT is the flagship task for NLP Deep Learning

- NMT research has pioneered many of the recent innovations of NLP Deep Learning
- In 2019: NMT research continues to thrive
  - Researchers have found many, many improvements to the “vanilla” seq2seq NMT system we have presented today
  - But one improvement is so integral that it is the new vanilla…

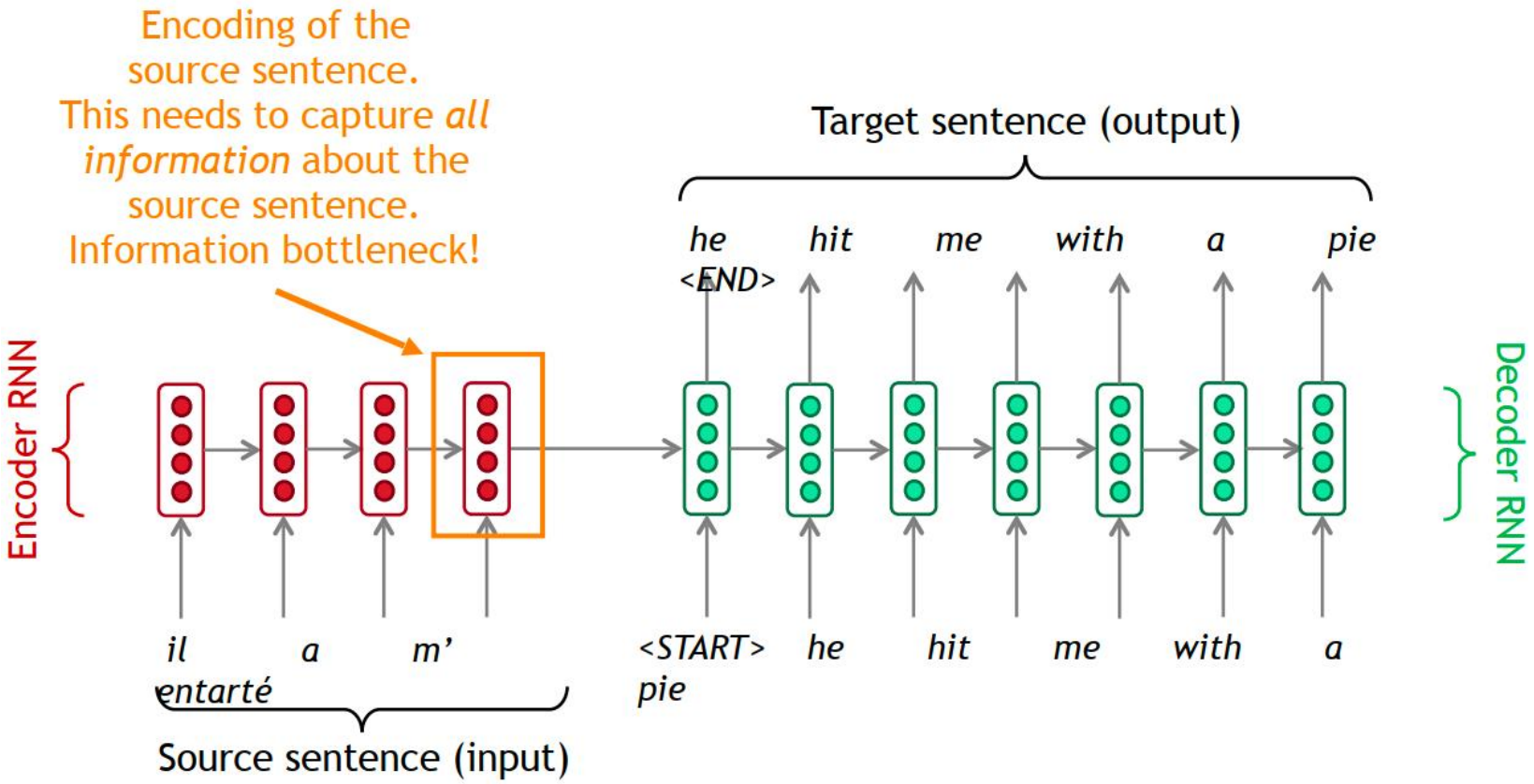
ATTENTION

Seq2Seq: The Bottleneck Problem



Problems with this architecture?

Seq2Seq: The Bottleneck Problem



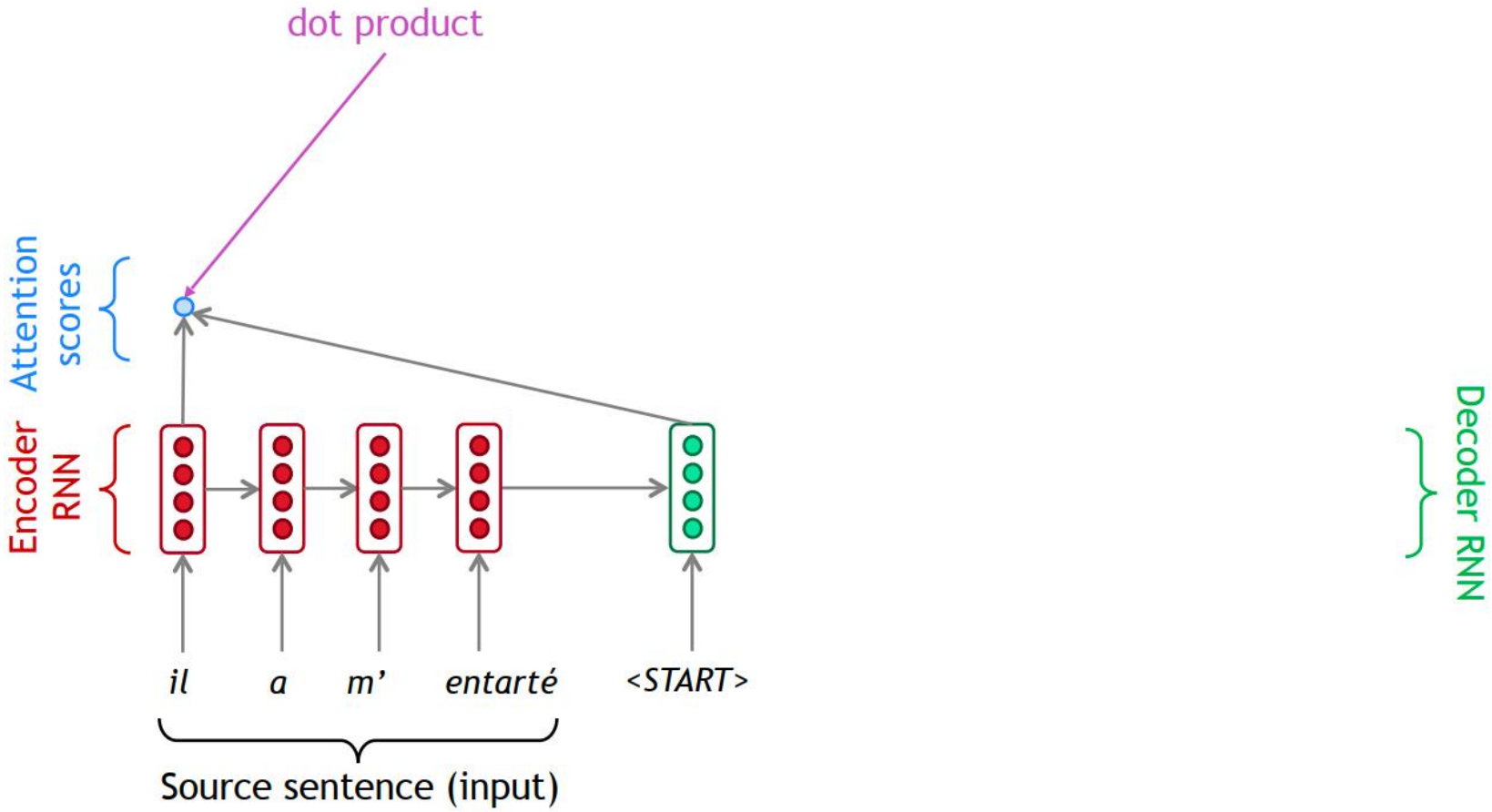
Attention

- Attention provides a solution to the bottleneck problem.

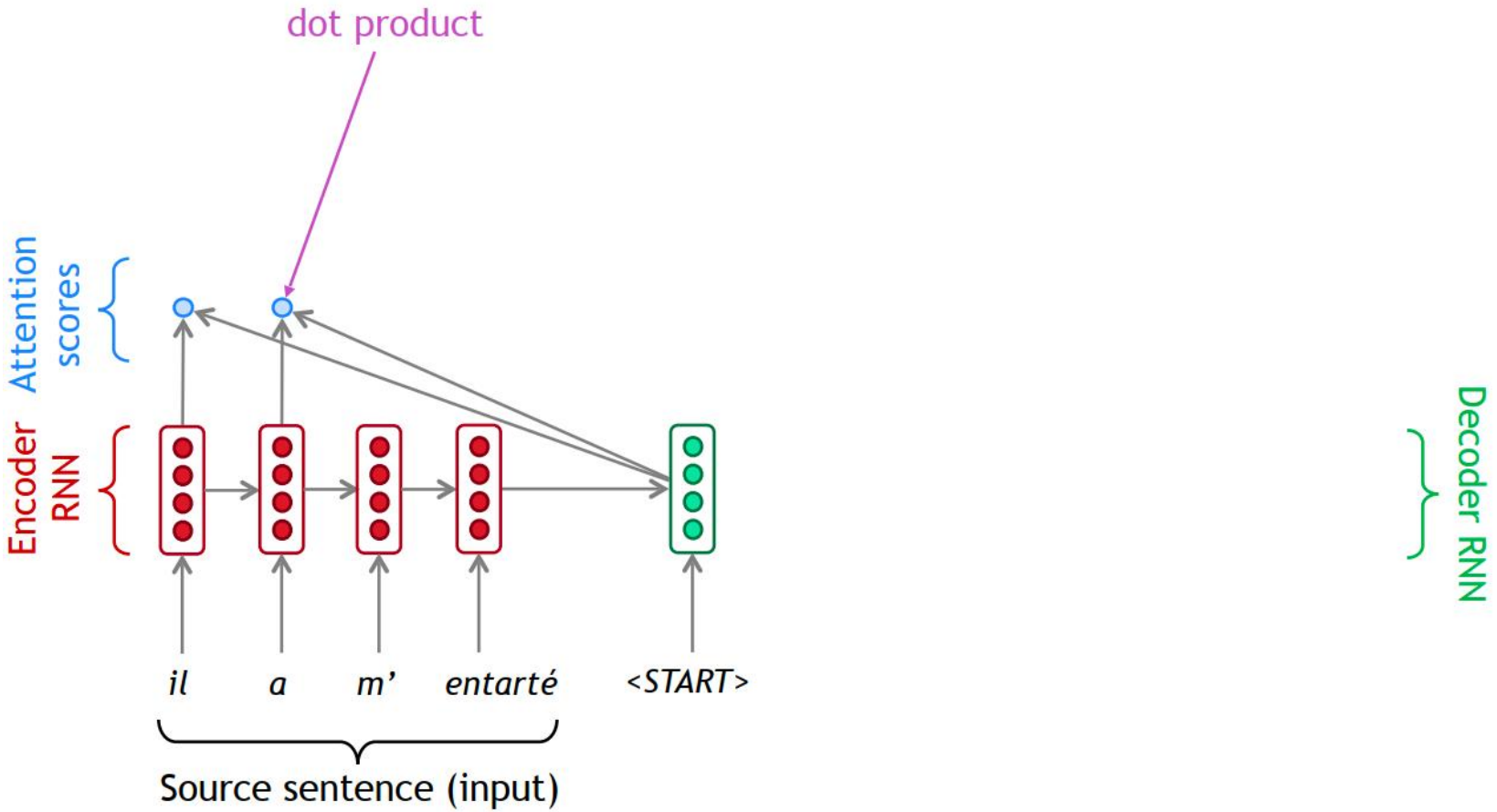
- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

- First, we will show via diagram (no equations), then we will show with equations

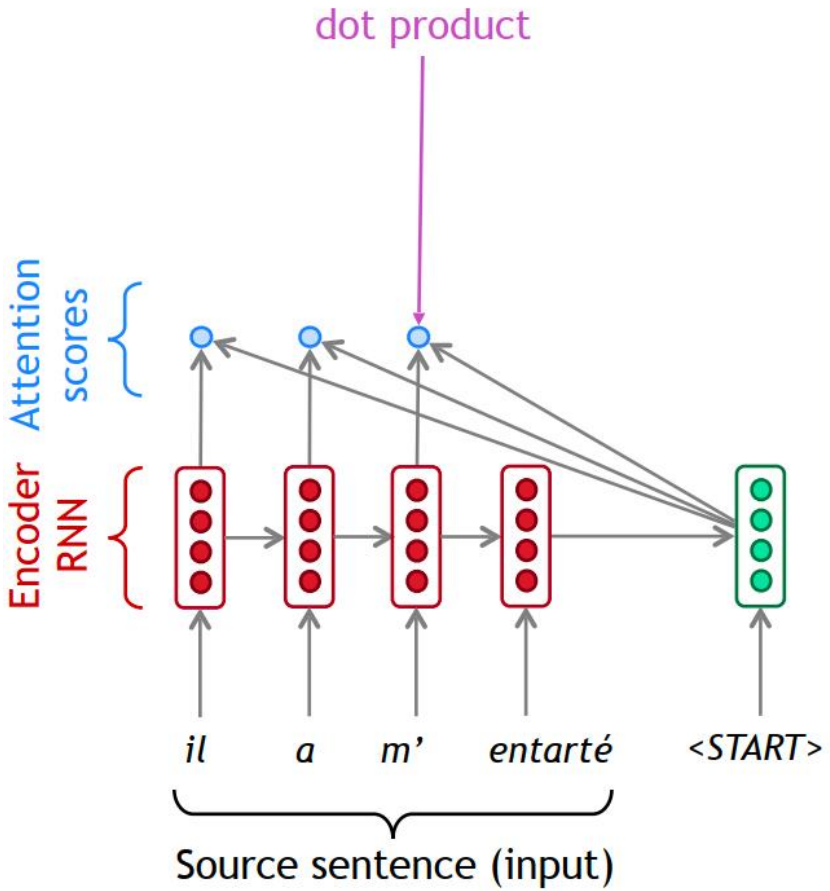
Seq2Seq with Attention



Seq2Seq with Attention

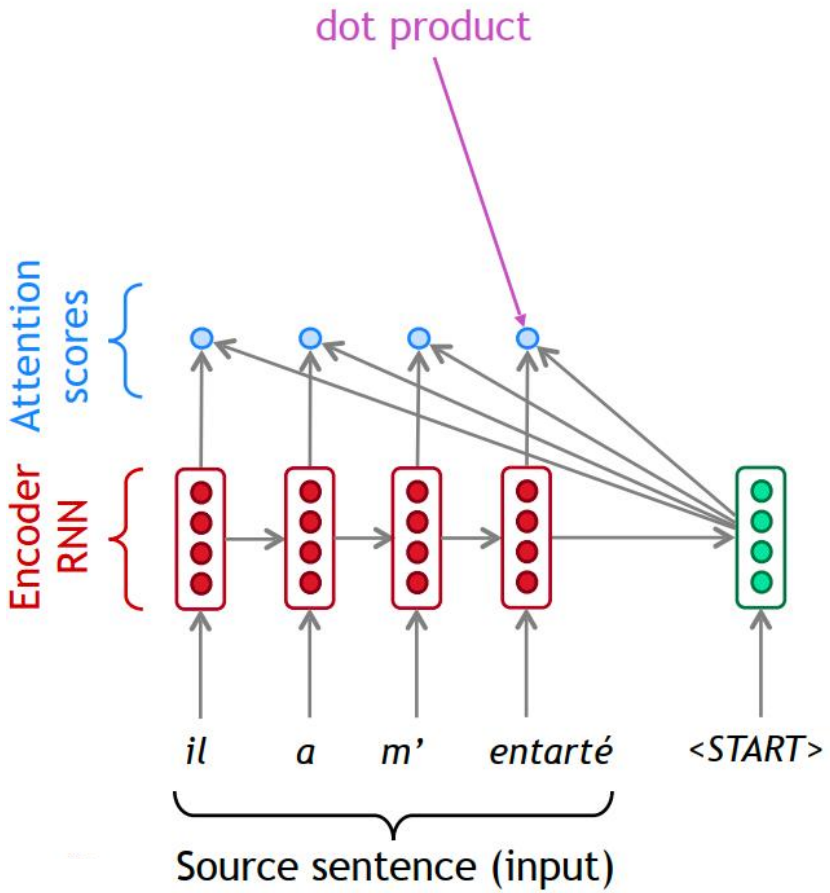


Seq2Seq with Attention



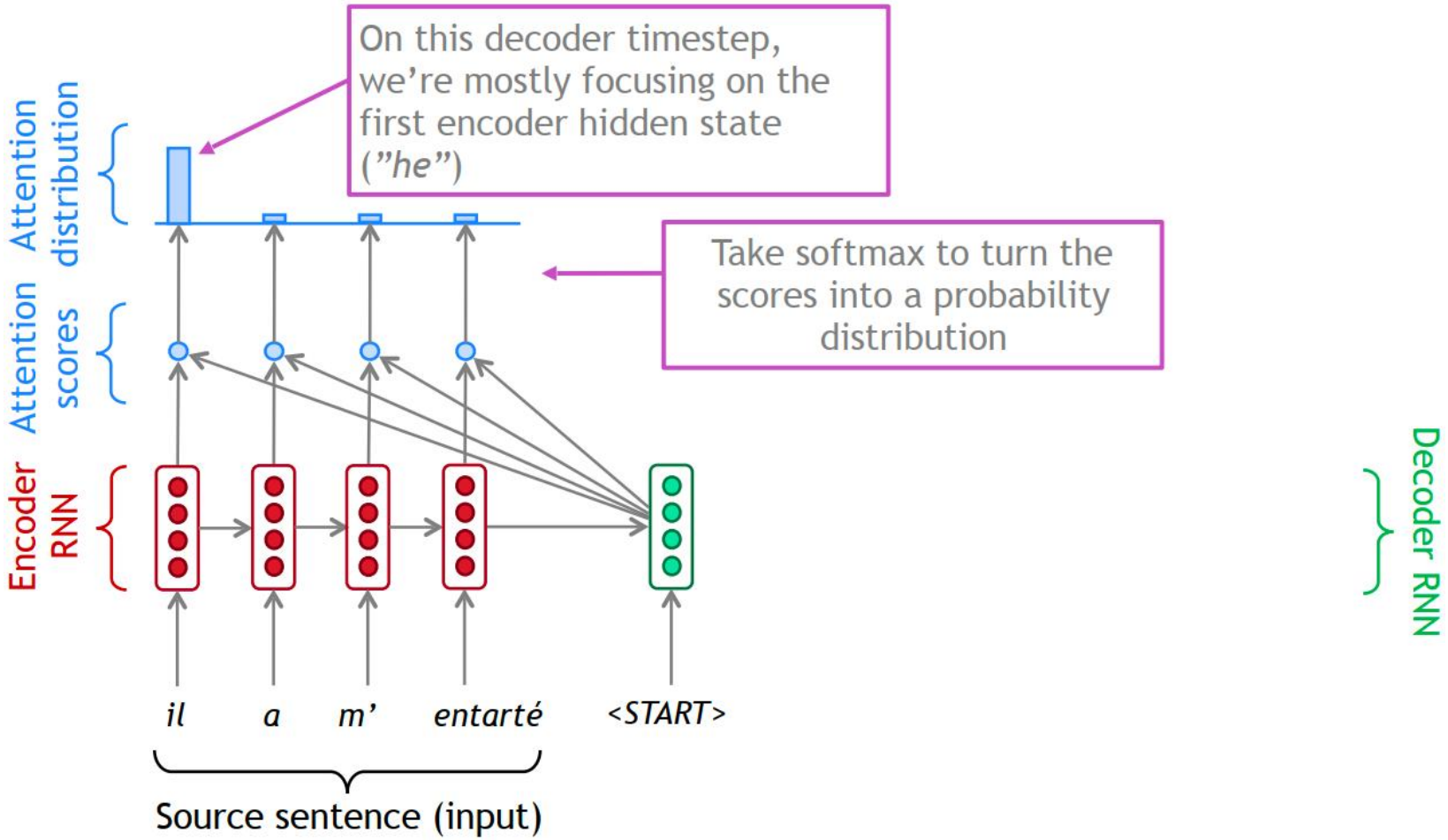
Decoder RNN

Seq2Seq with Attention

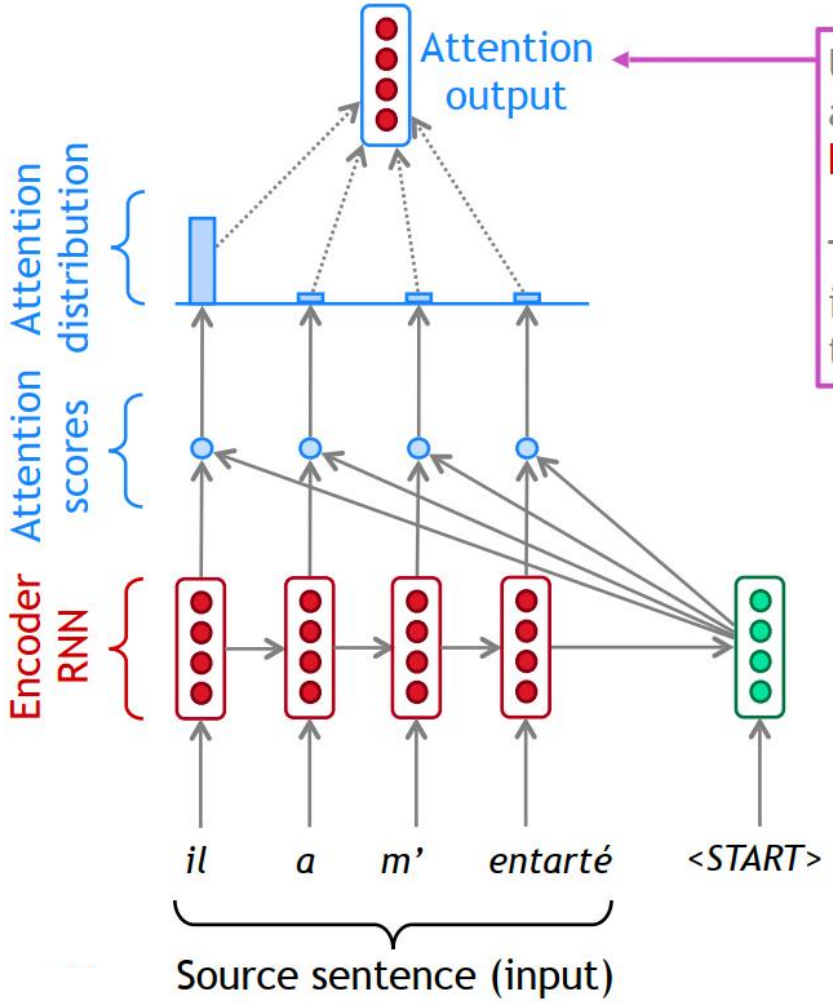


Decoder RNN

Sequence To Sequence



Sequence To Sequence

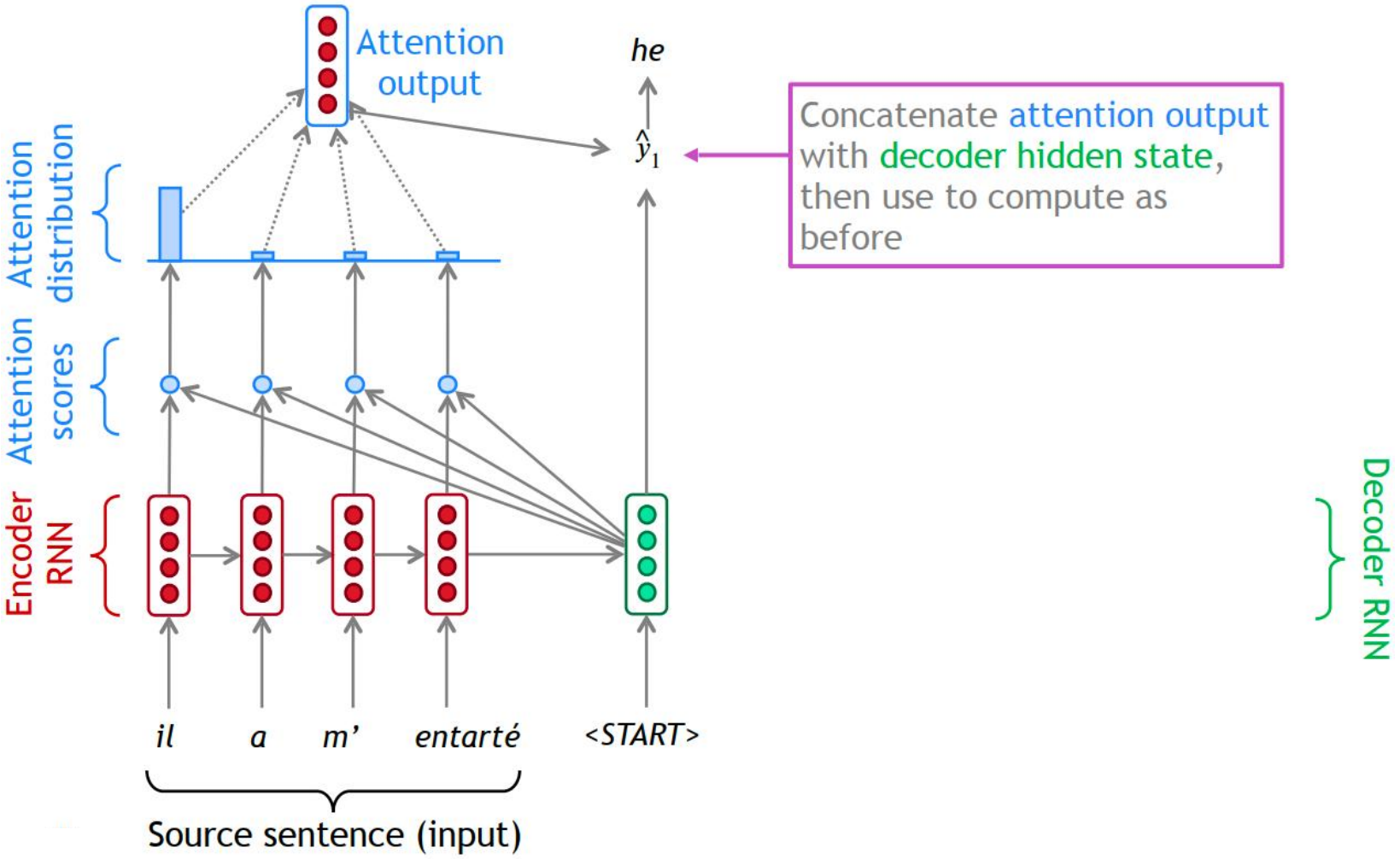


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

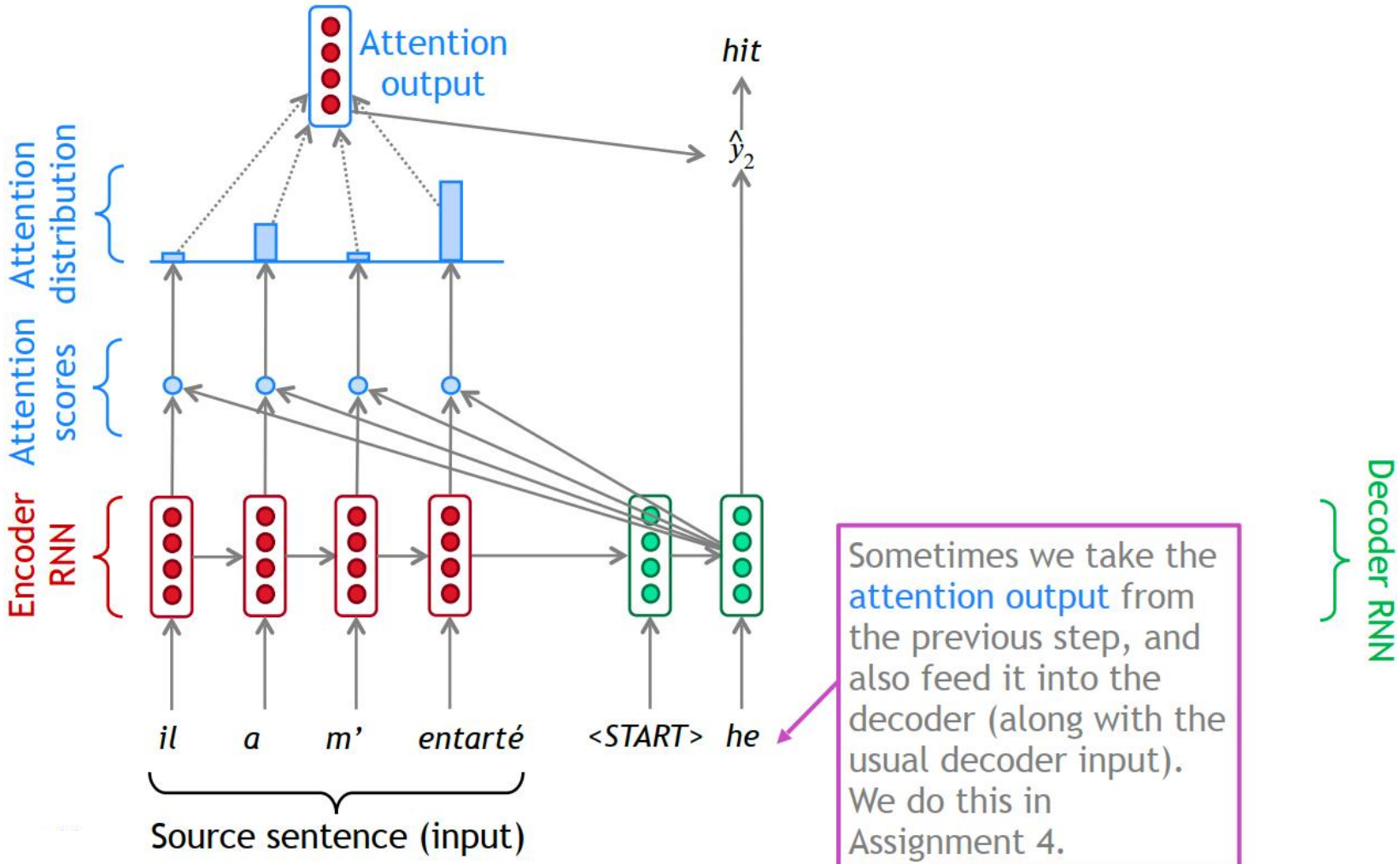
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

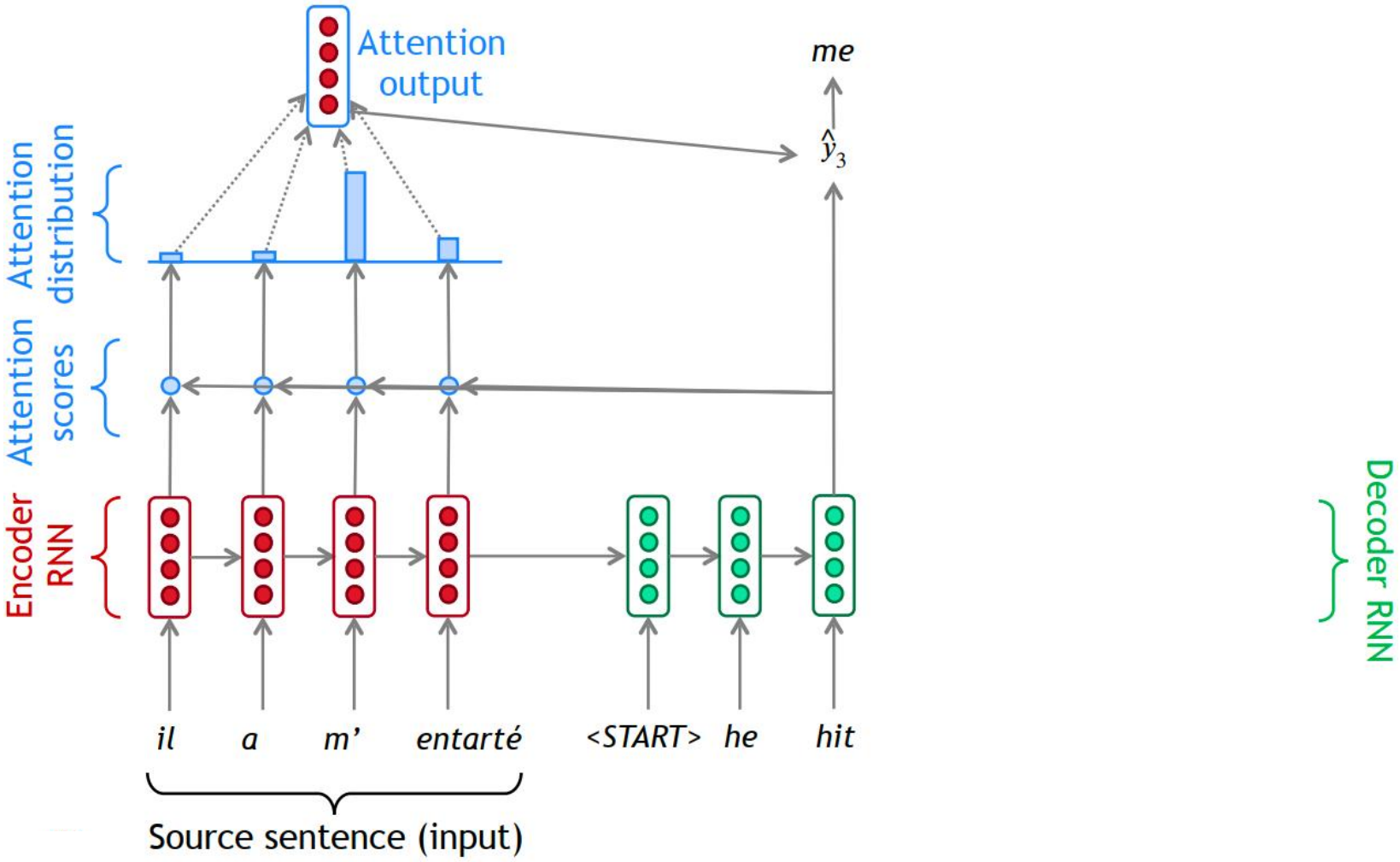
Sequence To Sequence



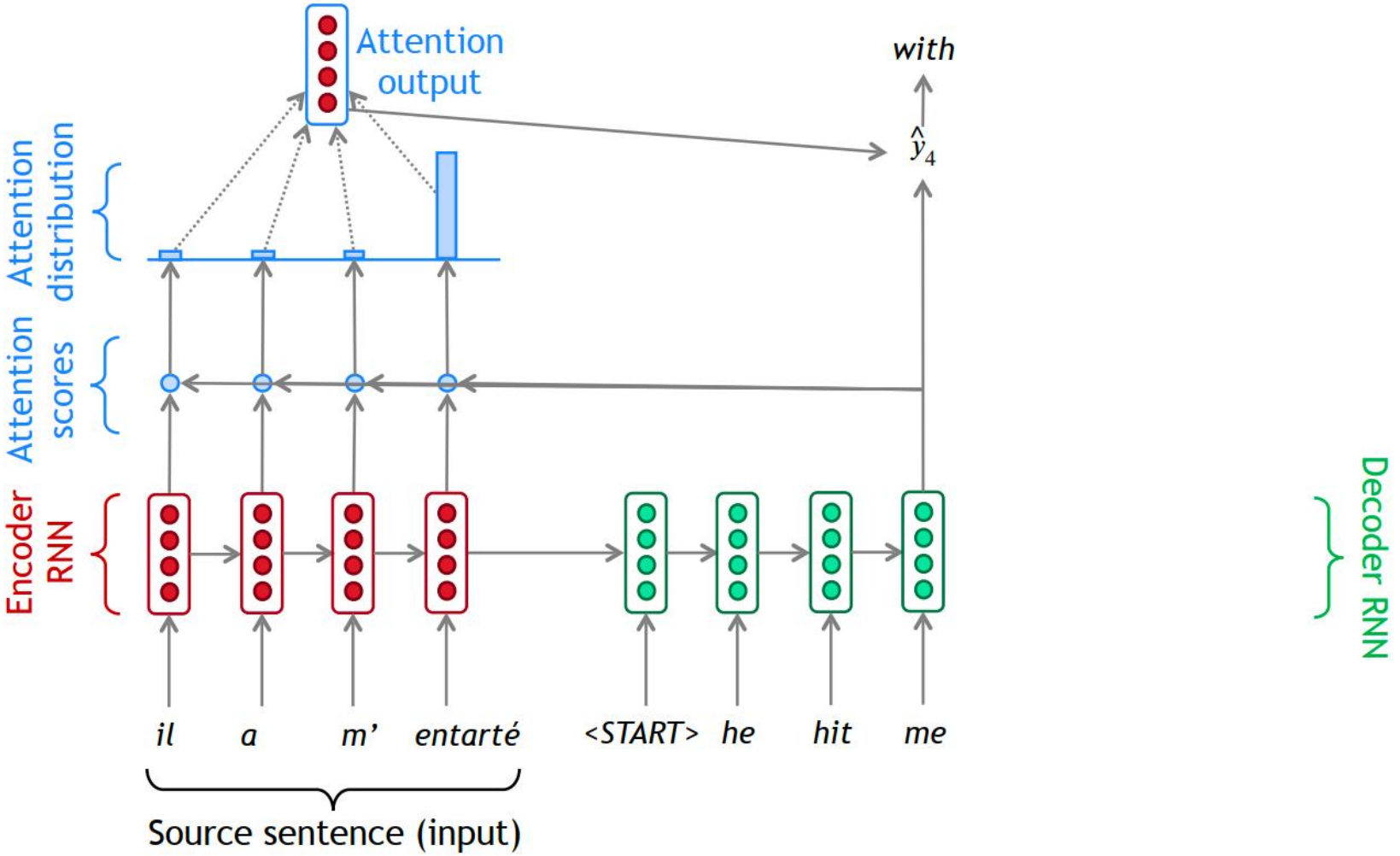
Sequence To Sequence



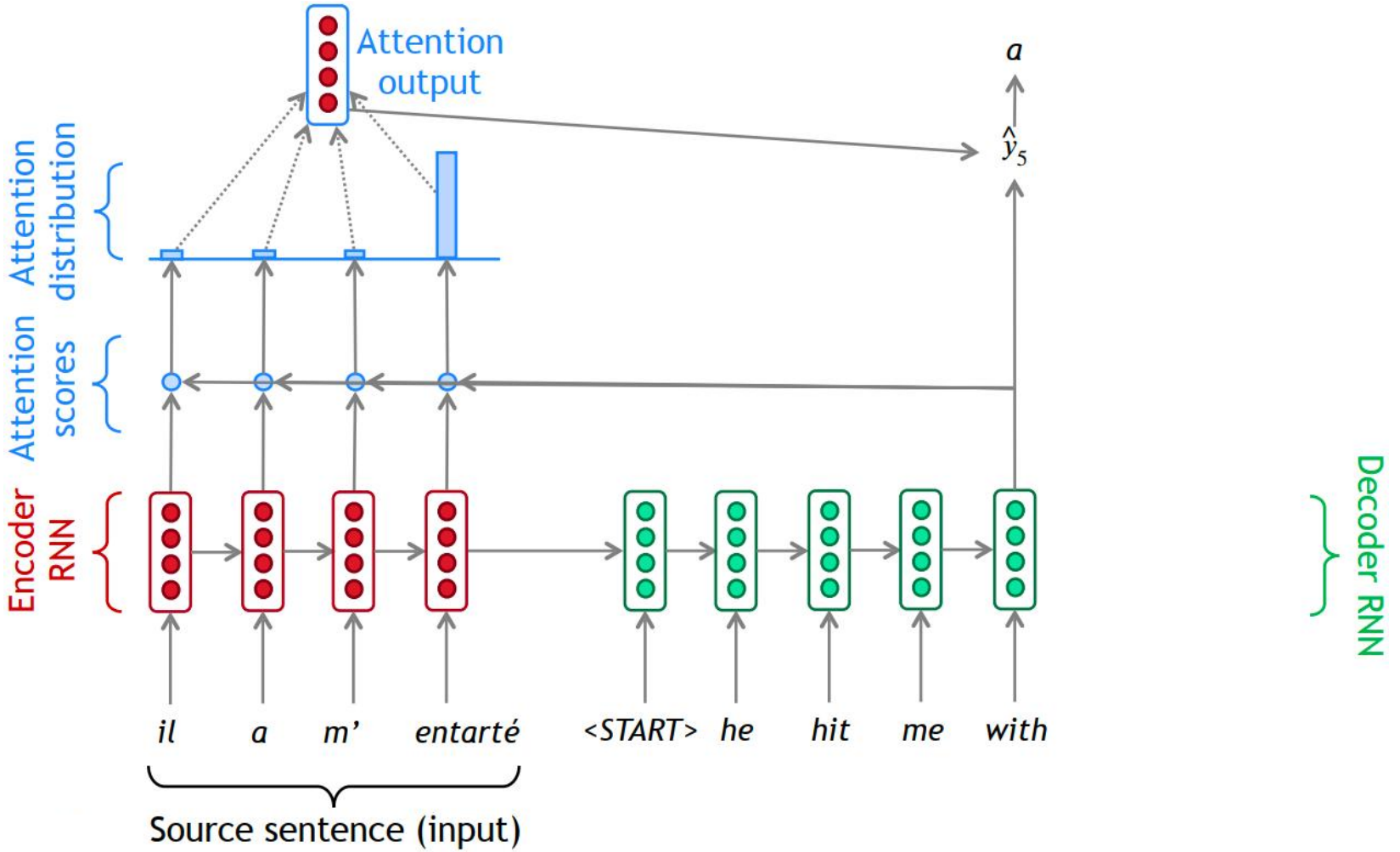
Sequence To Sequence



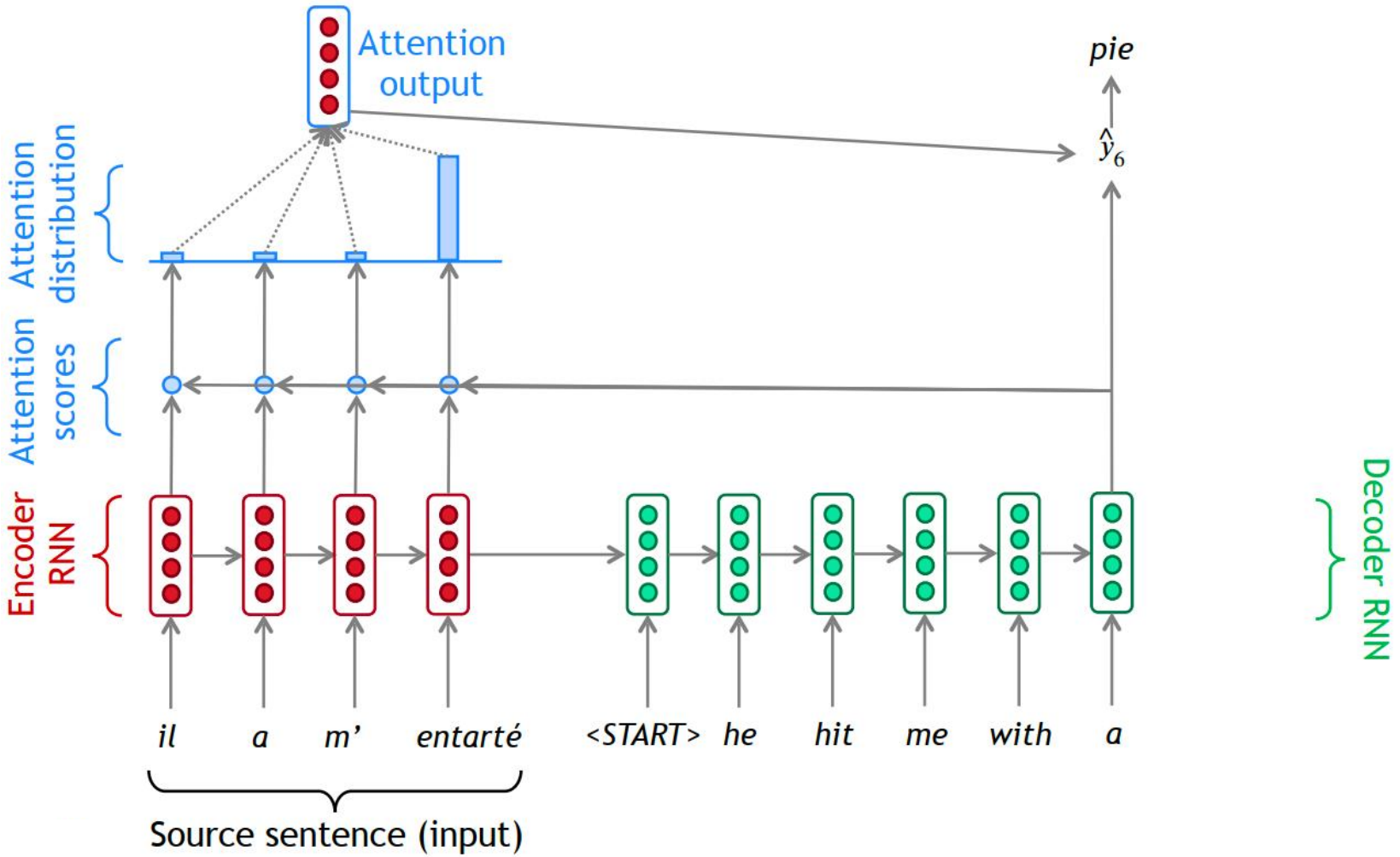
Sequence To Sequence



Sequence To Sequence



Sequence To Sequence



Attention: in equations

- We have encoder hidden states $h_1, ..., h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, ..., s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = softmax(e^t) \in \mathbb{R}^N$$

Attention: in equations

- We use a^t to take a weighted sum of the encoder hidden states to get the attention output: $a^t = \sum_{i=1}^{N} a_i^t \mathrm{h}_i \in \mathbb{R}^{\mathrm{h}}$
- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

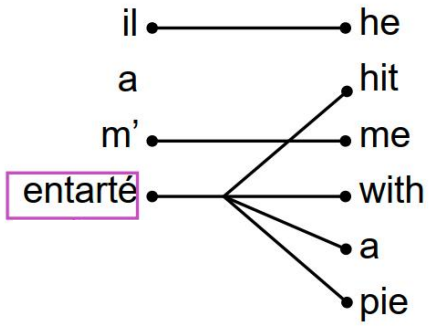
$$[a_t; s_t] \in \mathbb{R}^{2\mathrm{h}}$$

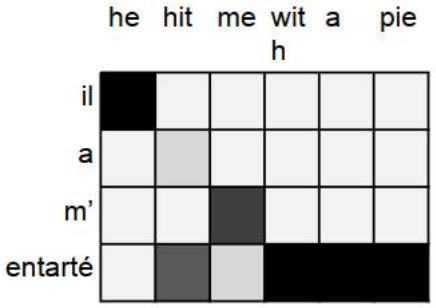
Attention is Great

- Attention significantly improves NMT performance
  - ✓ It is very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - ✓ Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - ✓ Provides shortcut to faraway states

Attention is Great

- Attention provides some interpretability
  - ✓ By inspecting attention distribution, we can see what the decoder was focusing on
  - ✓ We get (soft) alignment for free!
  - ✓ This is cool because we never explicitly trained an alignment system
  - ✓ The network just learned alignment by itself

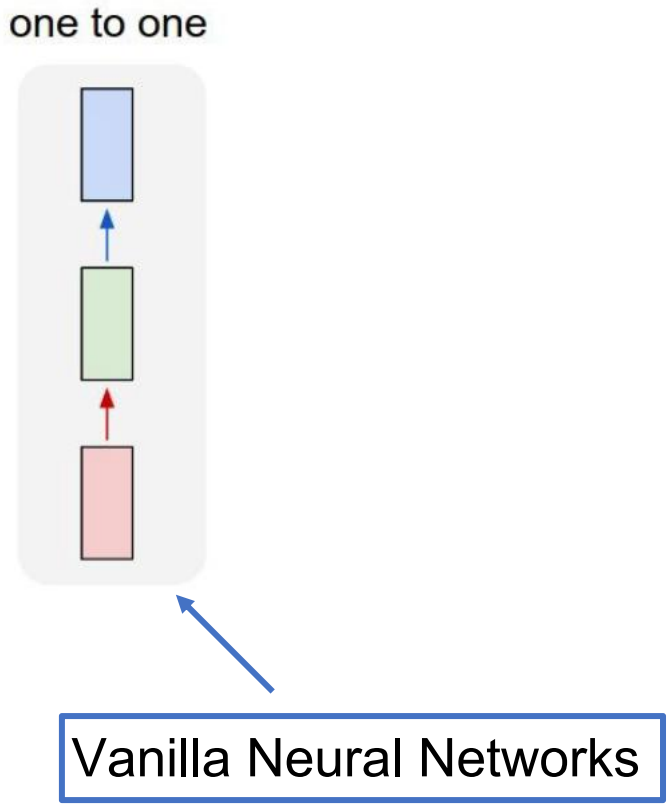




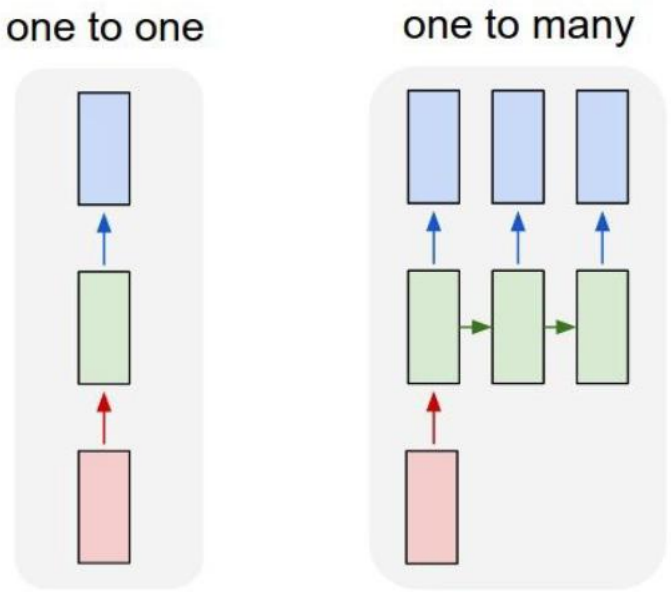
Attention Variants

- Basic dot-product attention: $e_i = s^T h_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier
- Multiplicative attention: $e_i = s^T W h_i \in \mathbb{R}$
  - Where $W \in \mathbb{R}^{d_1 \times d_2}$ is a weight matrix
- Additive attention: $e_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$
  - Where $W_1 \in \mathbb{R}^{d_3 \times d_1}$, $W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $v \in \mathbb{R}^{d_3}$ is a weight vector.
  - d_3 (the attention dimensionality) is a hyperparameter

Recurrent Neural Networks

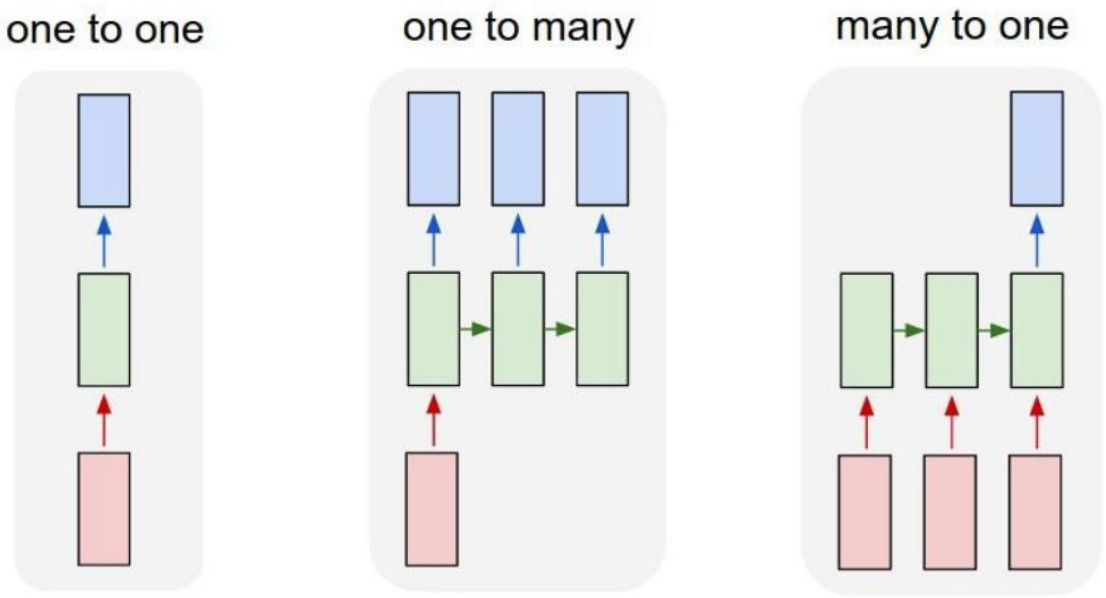


Recurrent Neural Networks



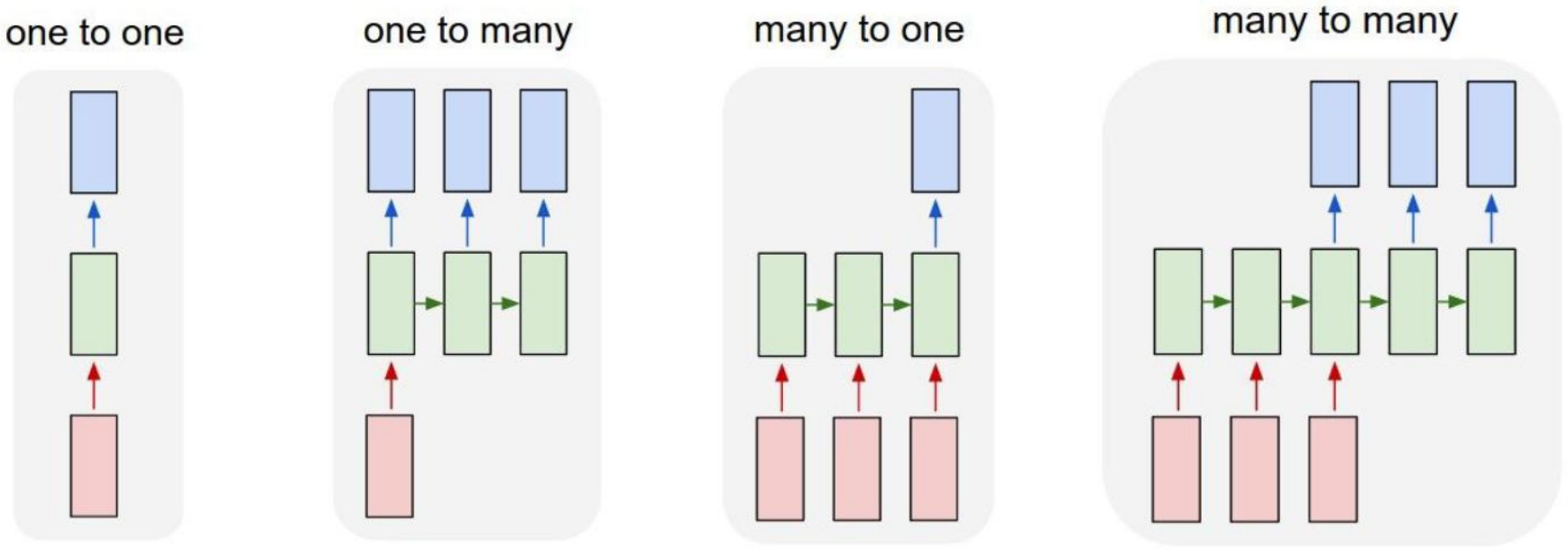
e.g. **Image Captioning**
Image -> Sequence of words

Recurrent Neural Networks



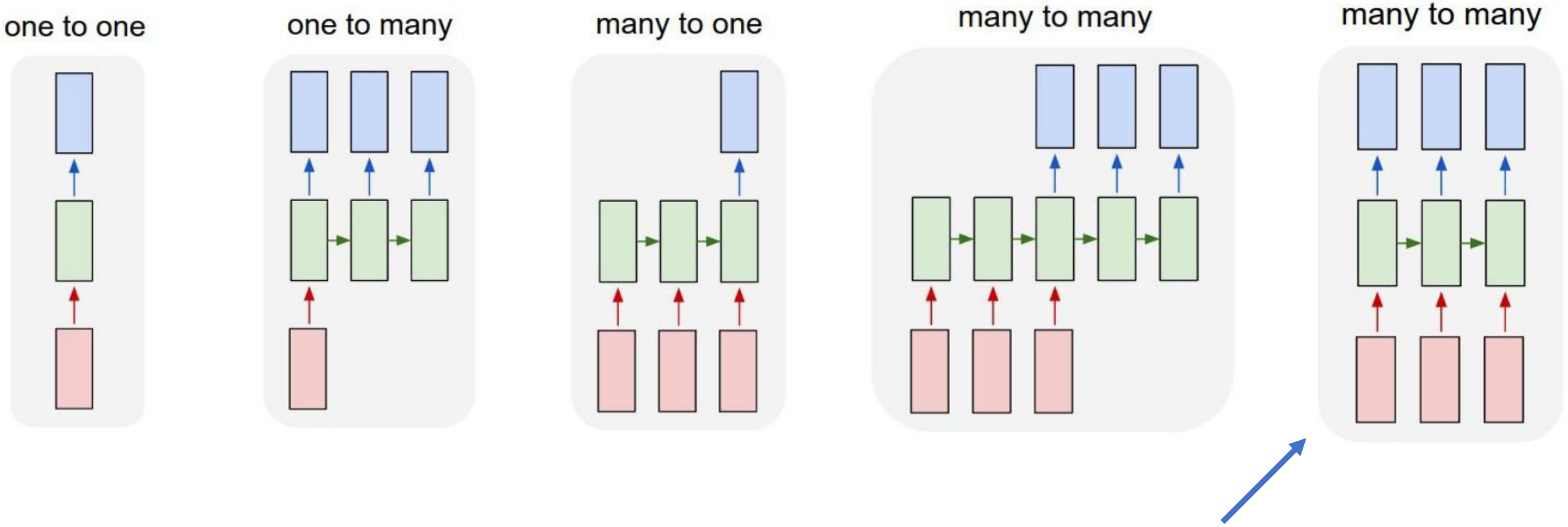
e.g. **Spam Detection**
Email -> Spam or not

Recurrent Neural Networks



e.g. **Video Caption**
Video -> Caption

Recurrent Neural Networks



e.g. **Video classification on frame level**

Problem of Sequential Data

- Single word Understanding is easy
  - *From previous word representations solution*
- But …… How to representing words by their context

Problem of Sequential Data

- Single word Understanding is easy
  - ✓ *From previous word representations solution*
- But …… How to representing words by their context
  - ✓ *Word Embedding Table is not enough*
  - ✓ *We need to model sequential information explicitly*

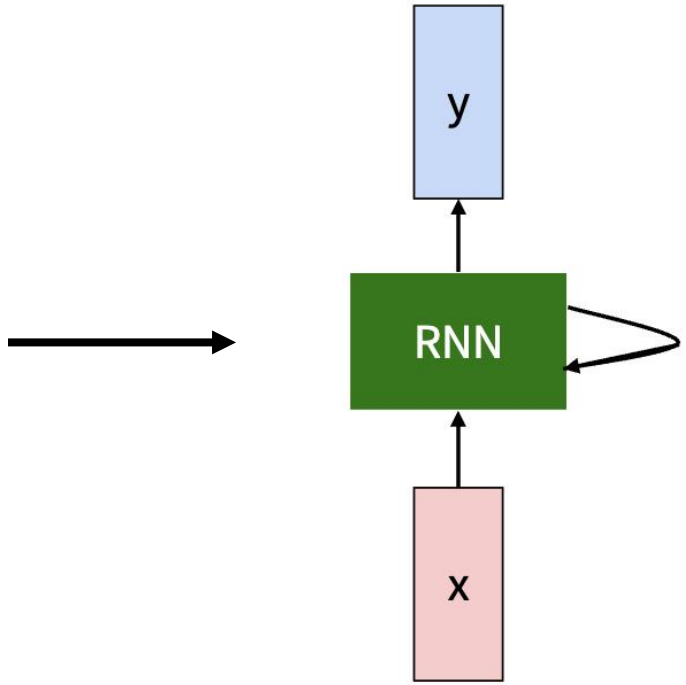
…government debt problems turning into **banking** crises as happened in 2009…
…saying that Europe needs unified **banking** regulation to replace the hodgepodge…
…India has just given its **banking** system a shot in the arm…

Different meanings of **banking** in different **context**

Theory and Computation

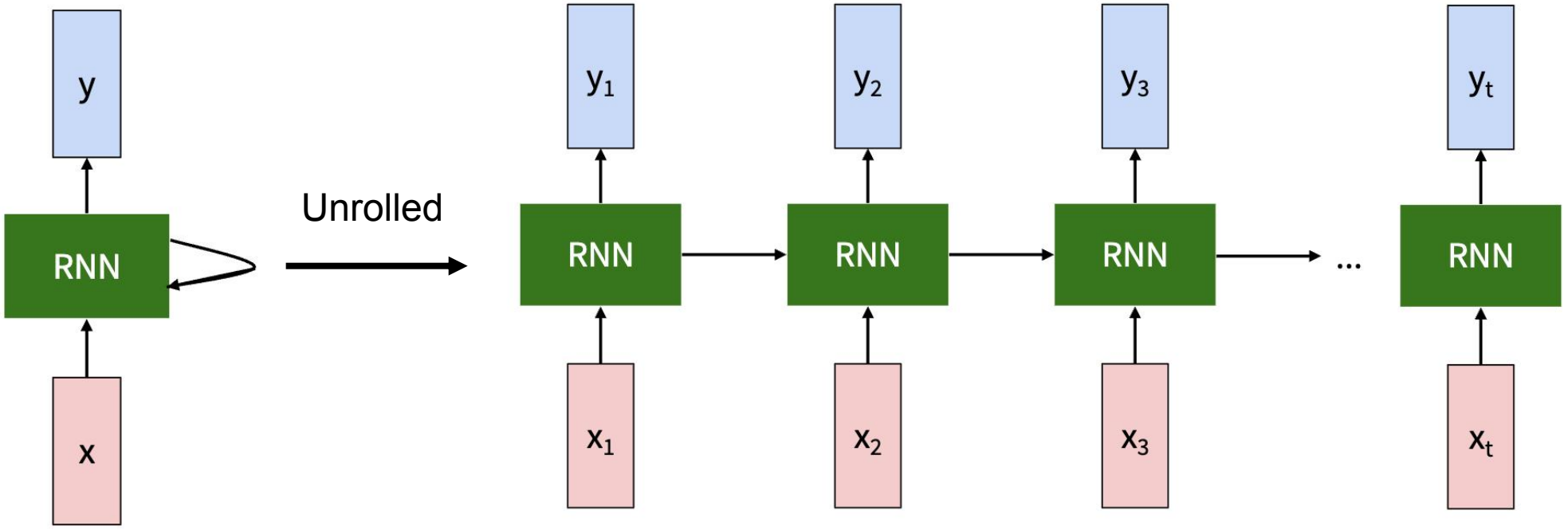
- To represent sequential information, model must have some "**memory mechanism**"

RNNs have an "internal state" that is updated as a sequence is processed



Theory and Computation

- Unrolled "**memory mechanism**": The hidden state from the previous step will be passed to the next step's input and used together as the input for the RNN.

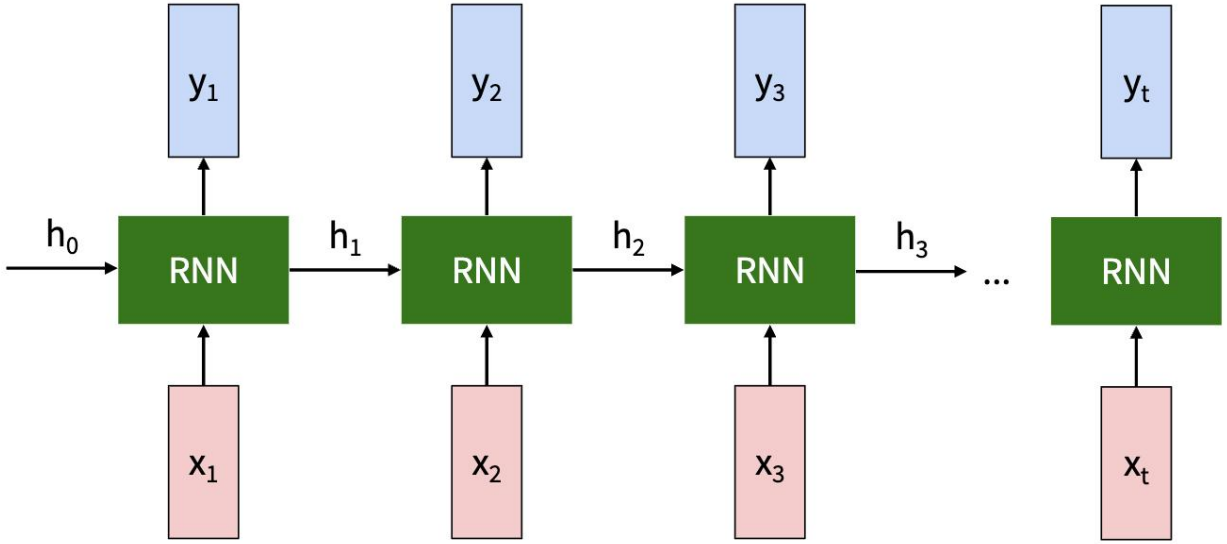


Theory and Computation

- Hidden State Update: $h_t = f_W(h_{t-1}, x_t)$

✓ h_t represents the hidden state at the current step, computed using the previous hidden state h_{t-1} and the current input x_t .

✓ The previous hidden state is passed to the next step, combined with the new input to update the RNN's state.



Theory and Computation

- Output Generation:

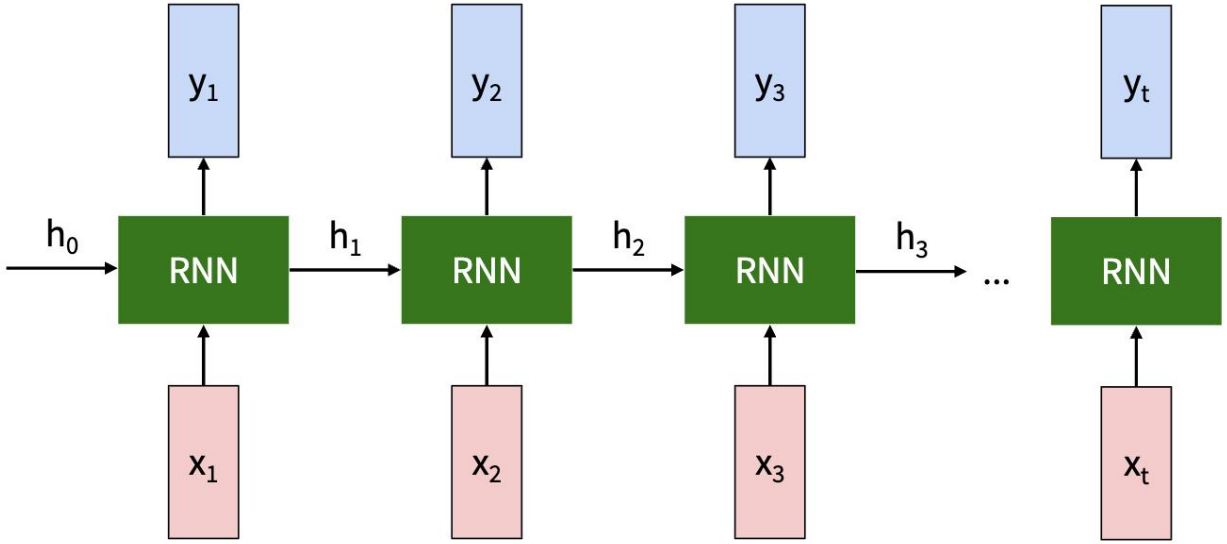
$$y_t = f_{W_{hy}}(h_t)$$

output

new state

another function with parameters W_{hy}

y_t represents the output in step t



Theory and Computation

- **Formula detail**: The same function and the same set of parameters (w, b) are used at every time step

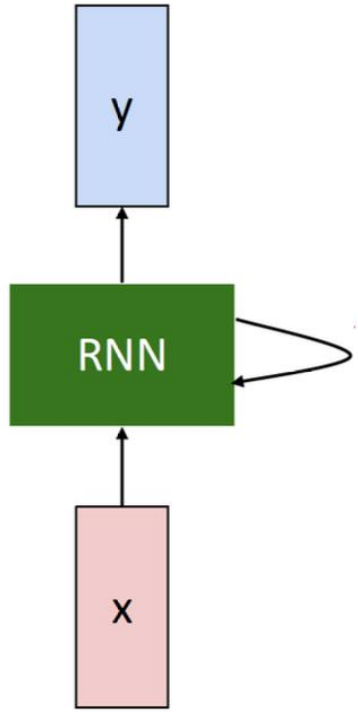
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

(Matmul then Add)

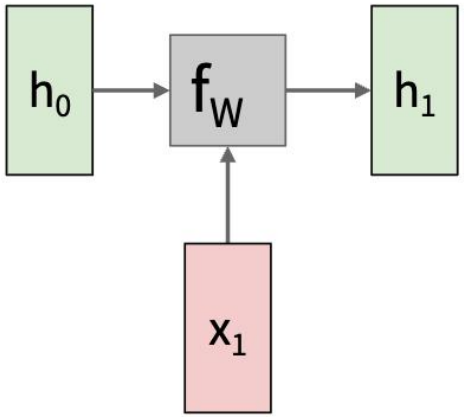
$$y_t = W_{hy}h_t + b_y$$

(Be like Vanilla NN)



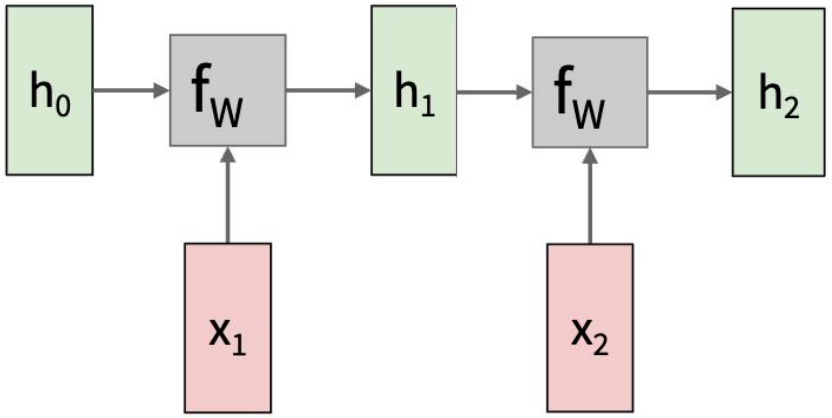
Theory and Computation

- The hidden state h_0 can be initiated randomly and learn



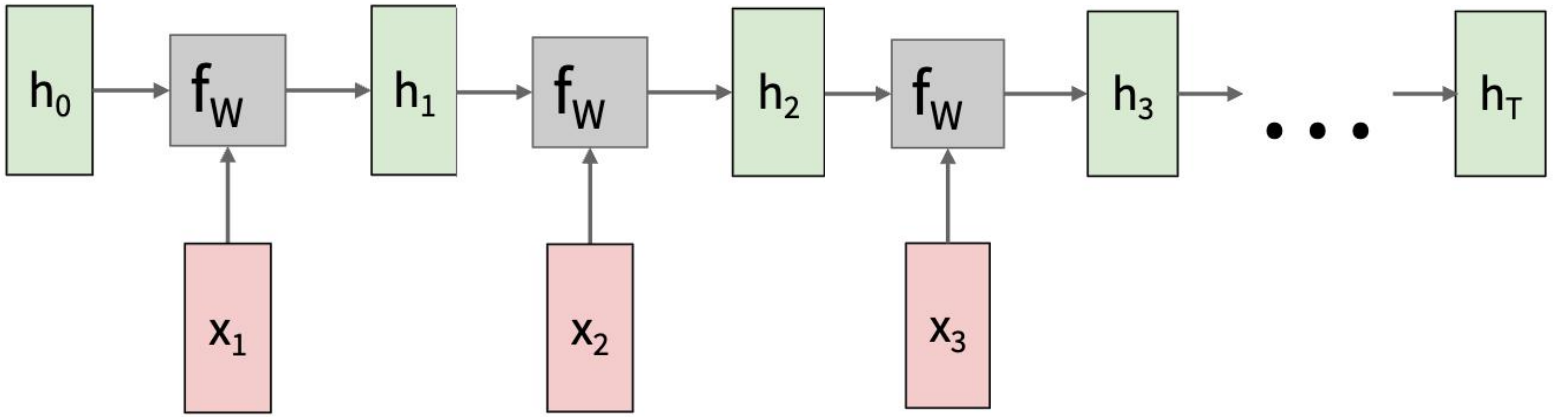
Theory and Computation

- The hidden state h_0 can be initiated randomly and learn



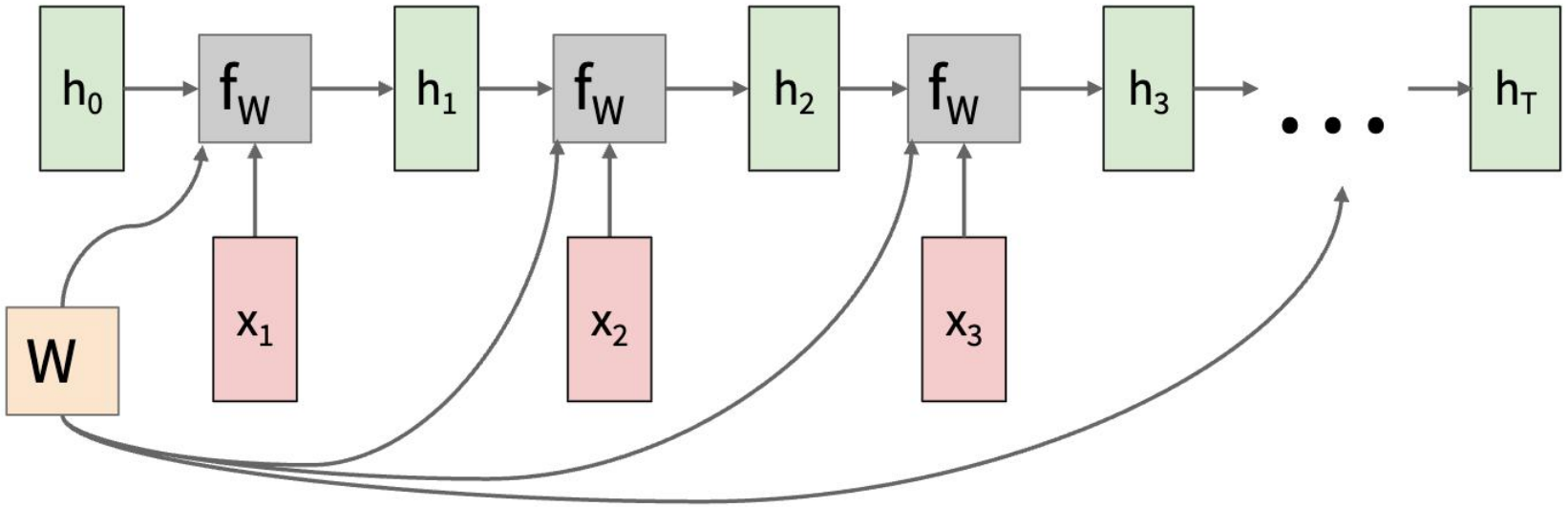
Theory and Computation

- The hidden state h_0 can be initiated randomly and learn



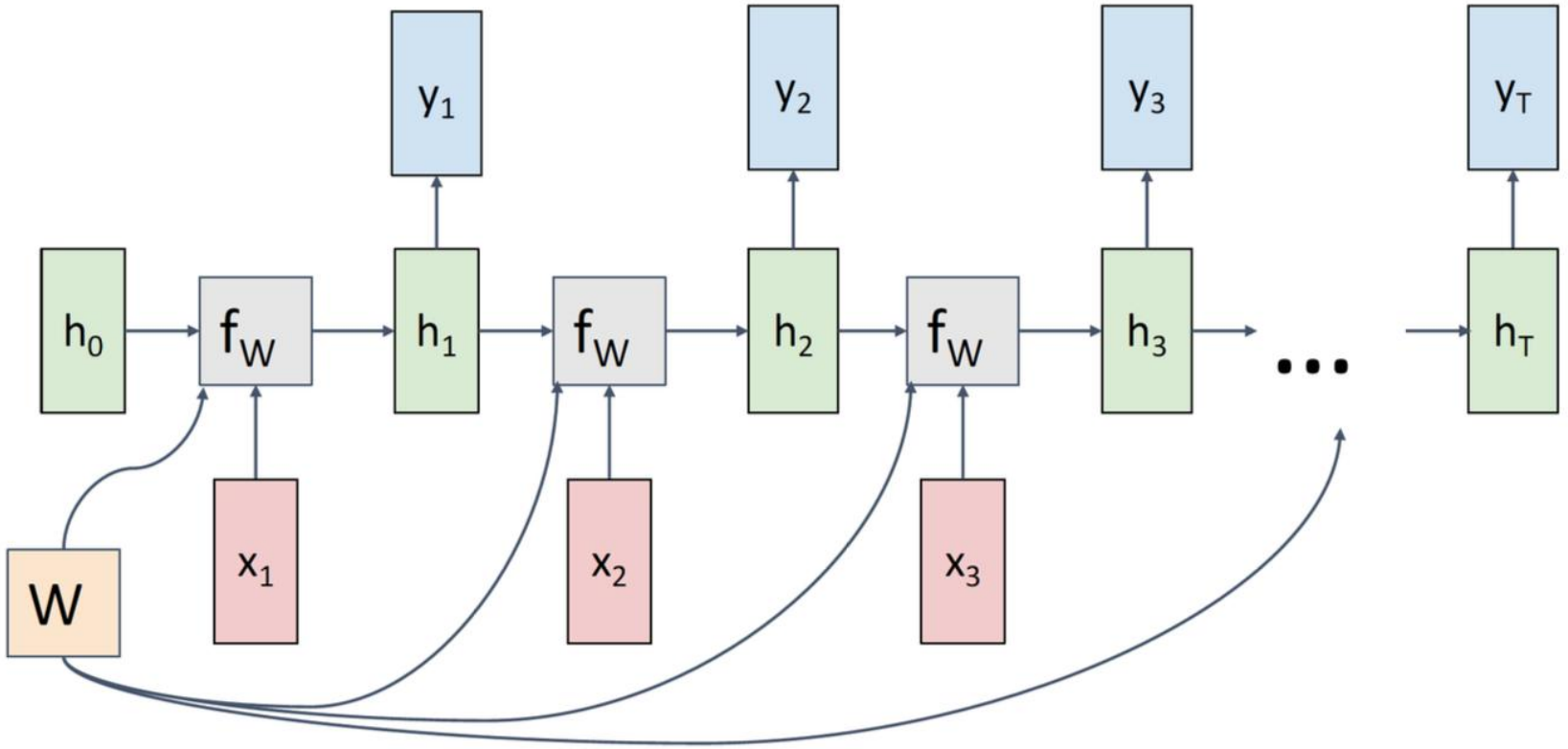
Theory and Computation

- The same function and the same set of parameters (w, b) are used at every time step



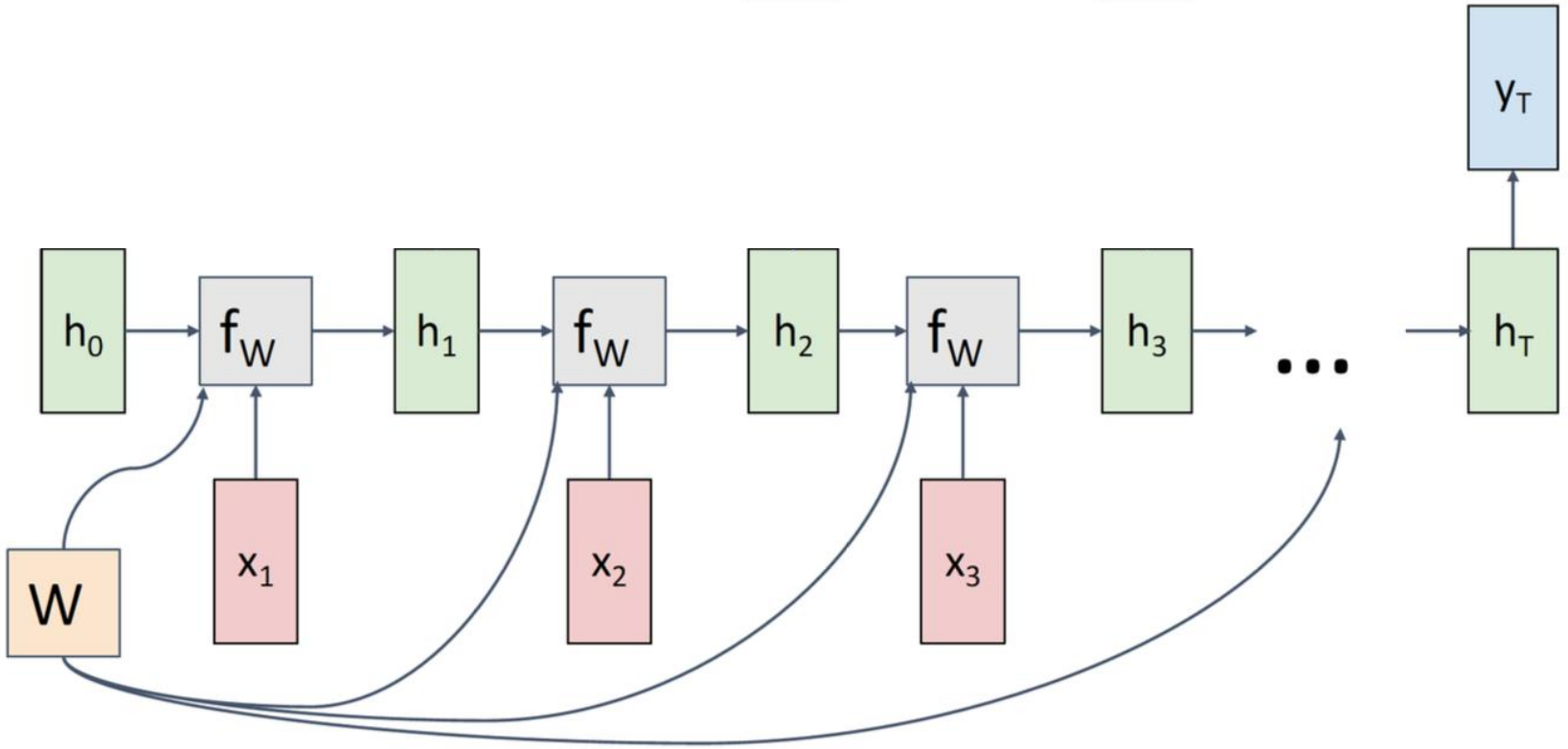
Theory and Computation

- Many to Many: e.g. Machine Translation



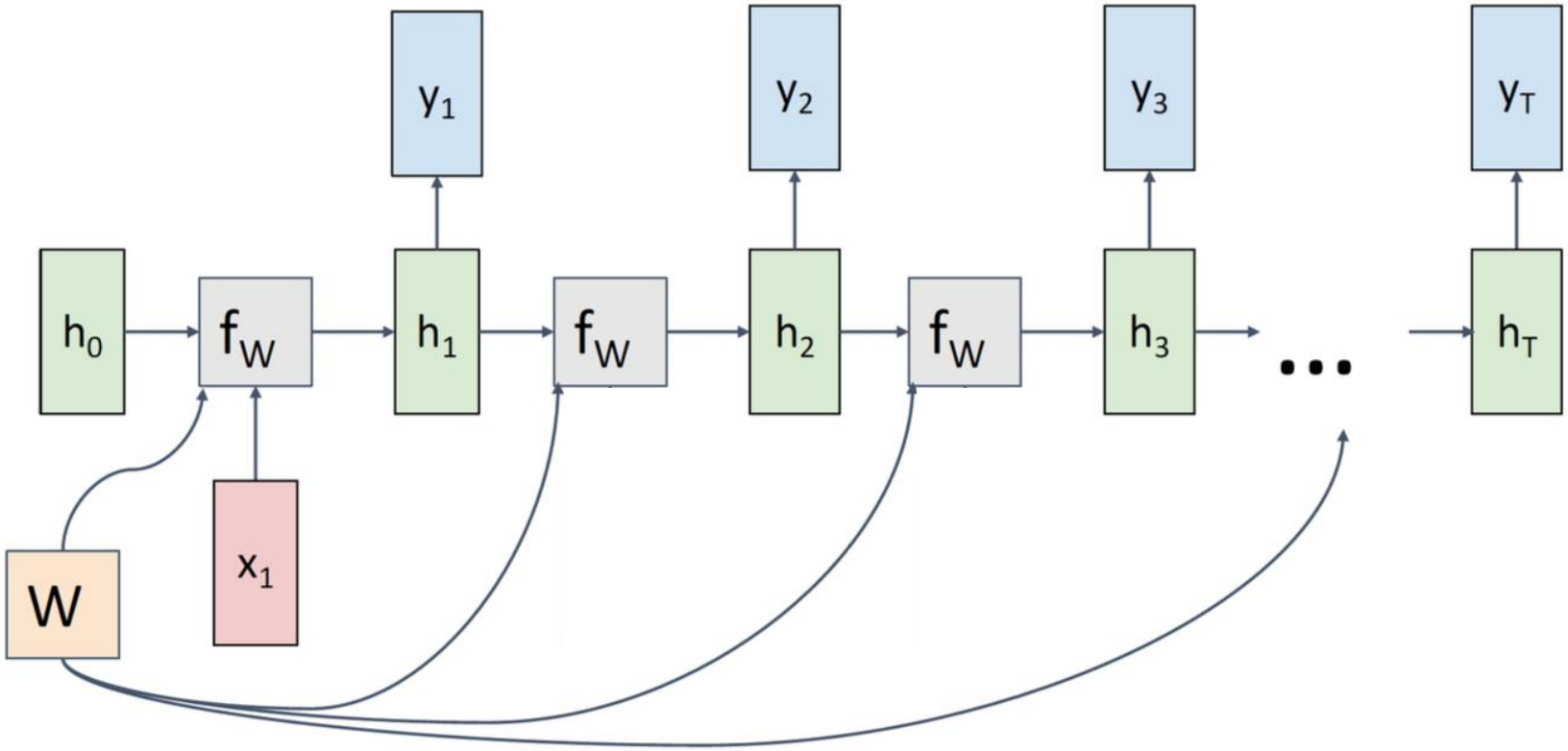
Theory and Computation

- Many to One: e.g. Spam Detection



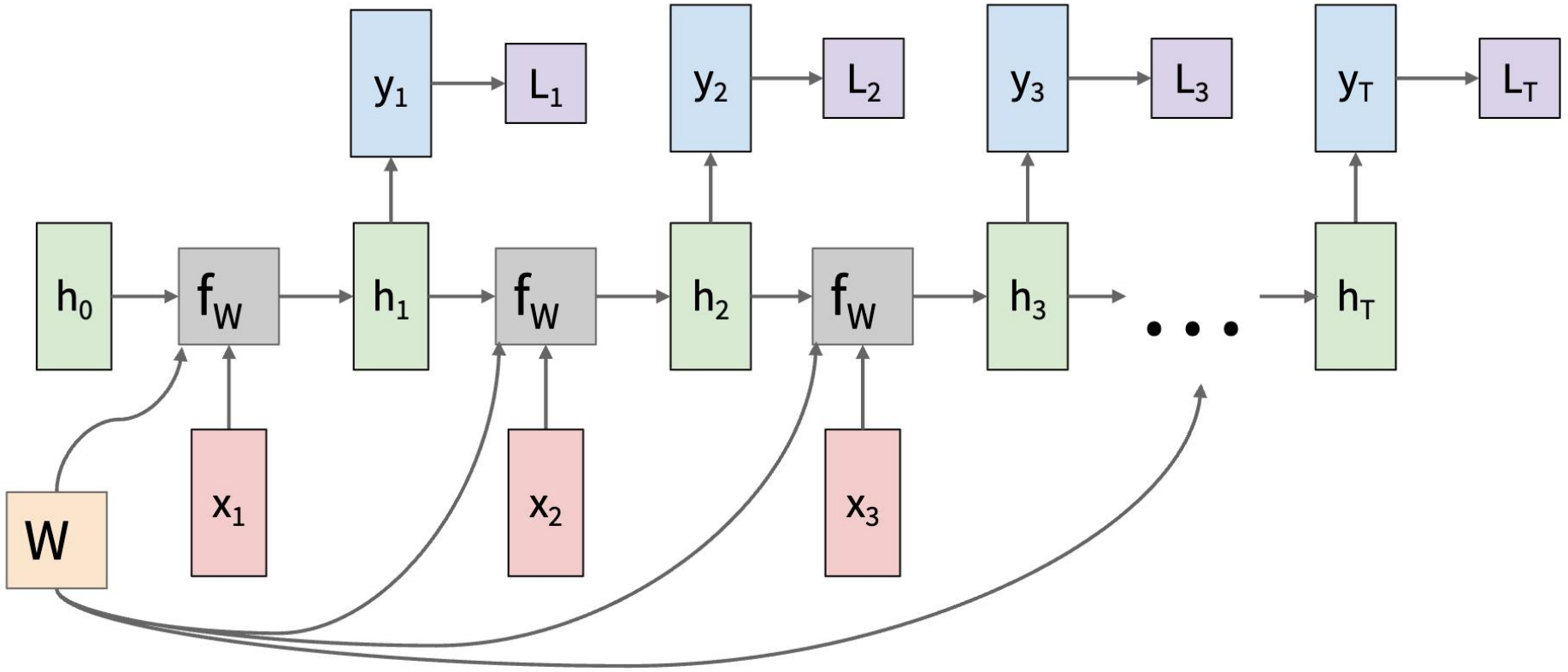
Theory and Computation

- One to Many: e.g. Image caption



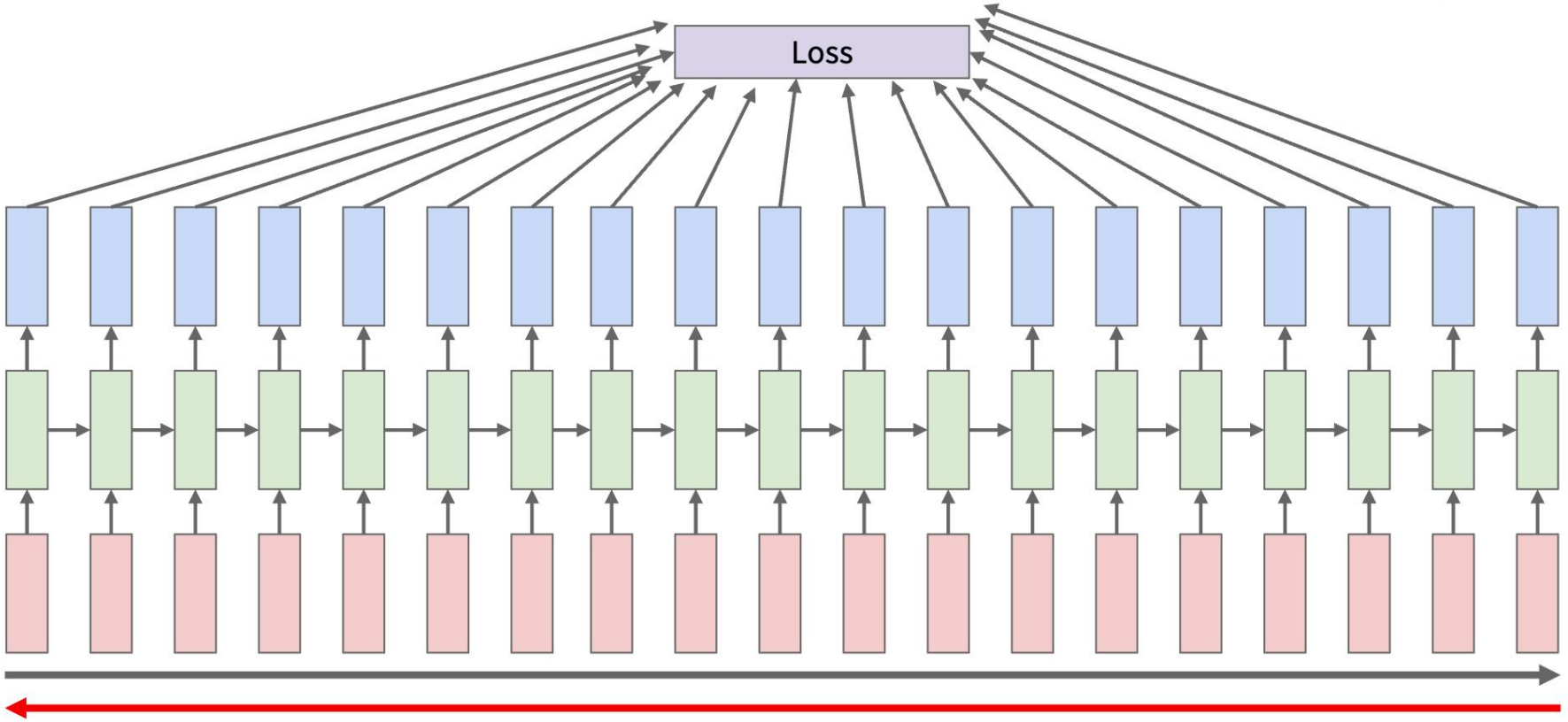
Theory and Computation

- **Training**: We can compare the prediction y_t with Ground truth x_{t+1} to compute the loss L_t



Theory and Computation

- Training Strategy: Back propagation through time



Thanks

Welcome to join VisionXLab

yangxue.site